Use of Hybrid Public-Key Encryption (HPKE) with CBOR Object Signing and Encryption (COSE)
draft-ietf-cose-hpke-01

Abstract

This specification defines hybrid public-key encryption (HPKE) for use with CBOR Object Signing and Encryption (COSE). HPKE offers a variant of public-key encryption of arbitrary-sized plaintexts for a recipient public key.

HPKE works for any combination of an asymmetric key encapsulation mechanism (KEM), key derivation function (KDF), and authenticated encryption with additional data (AEAD) encryption function. Authentication for HPKE in COSE is provided by COSE-native security mechanisms.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction ........................................... 3
2. Conventions and Terminology .......................... 3
3. HPKE for COSE ........................................ 3
   3.1. Overview ........................................ 3
   3.2. HPKE Encryption with SealBase .................... 4
   3.3. HPKE Decryption with OpenBase .................... 5
   3.4. Info Structure .................................. 5
4. Example ................................................ 6
5. Security Considerations ............................... 7
6. IANA Considerations ................................... 8
   6.1. HPKE/P-256+HKDF-256 and AES-128-GCM ............. 8
   6.2. HPKE/P-512+HKDF-512 and AES-256-GCM ............. 8
7. References ............................................ 8
   7.1. Normative References .............................. 9
   7.2. Informative References ............................ 9
Appendix A. Acknowledgements ............................ 9
Authors’ Addresses ....................................... 9

1. Introduction

Hybrid public-key encryption (HPKE) [I-D.irtf-cfrg-hpke] is a scheme that provides public key encryption of arbitrary-sized plaintexts given a recipient’s public key. HPKE utilizes a non-interactive ephemeral-static Diffie-Hellman exchange to establish a shared secret. The motivation for standardizing a public key encryption scheme is explained in the introduction of [I-D.irtf-cfrg-hpke].

The HPKE specification defines several features for use with public key encryption and a subset of those features is applied to COSE [RFC8152]. Since COSE provides constructs for authentication, those are not re-used from the HPKE specification. This specification uses the "base" mode, as it is called in HPKE specification language.

2. Conventions and 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.

This specification uses the following abbreviations and terms:
- Content-encryption key (CEK), a term defined in CMS [RFC2630].
- Hybrid Public Key Encryption (HPKE) is defined in [I-D.irtf-cfrg-hpke].
- pkR is the public key of the recipient, as defined in [I-D.irtf-cfrg-hpke].
- skR is the private key of the recipient, as defined in [I-D.irtf-cfrg-hpke].

3. HPKE for COSE

3.1. Overview

The CDDL for the COSE_Encrypt structure, as used with this specification, is shown in Figure 1.

HPKE, when used with COSE, follows a two layer structure:

* Layer 0 (corresponding to the COSE_Encrypt structure) contains content (plaintext) encrypted with the CEK. This ciphertext may be detached. If not detached, then it is included in the COSE_Encrypt structure.

* Layer 1 (corresponding to a recipient structure) contains parameters needed for HPKE to generate a shared secret used to encrypt the CEK. This layer includes the encrypted CEK.
This two-layer structure is used to encrypt content that can also be shared with multiple parties at the expense of a single additional encryption operation. As stated above, the specification uses a CEK to encrypt the content at layer 0.

For example, the content encrypted at layer 0 is a firmware image. The same ciphertext firmware image is processed by all of the recipients; however, each recipient uses their own private key to obtain the CEK.

COSE_Encrypt_Tagged = #6.96(COSE_Encrypt)

HPKE_Encryption_Info = COSE_Encrypt_Tagged

; Layer 0
COSE_Encrypt = [
  Headers,
  ciphertext : bstr / nil,
  recipients : + COSE_recipient
]

; Layer 1
COSE_recipient = [
  protected : bstr .cbor header_map, ; must contain alg parameter
  unprotected : header_map, ; must contain kid and ephemeral public key
  encCEK : bstr, ; CEK encrypted with HPKE-derived shared secret
]

header_map = {
  Generic_Headers,
  * label =values,
}

Figure 1: CDDL for HPKE-based COSE_Encrypt Structure

The COSE_recipient structure shown in Figure 1 is repeated for each recipient, and it includes the encrypted CEK as well as the sender-generated ephemeral public key in the unprotected header structure.

3.2. HPKE Encryption with SealBase

The SealBase(pkR, info, aad, pt) function is used to encrypt a plaintext pt to a recipient’s public key (pkR).
IMPORTANT: For use in this specification, the plaintext "pt" passed into the SealBase is the CEK. The CEK is a random byte sequence of length appropriate for the encryption algorithm selected in layer 0. For example, AES-128-GCM requires a 16 byte key and the CEK would therefore be 16 bytes long.

The "info" parameter can be used to influence the generation of keys and the "aad" parameter provides additional authenticated data to the AEAD algorithm in use. This specification does not mandate the use of the info and the aad parameters.

If SealBase() is successful, it will output a ciphertext "ct" and an encapsulated key "enc". The content of enc is the ephemeral public key.

The content of the info parameter is based on the 'COSE_KDF_Context' structure, which is detailed in Figure 2.

3.3. HPKE Decryption with OpenBase

The recipient will use the OpenBase(enc, skR, info, aad, ct) function with the enc and ct parameters received from the sender. The "aad" and the "info" parameters are obtained via the context of the usage.

The OpenBase function will, if successful, decrypt "ct". When decrypted, the result will be the CEK. The CK is the symmetric key used to decrypt the ciphertext in layer 0 of the COSE_Encrypt structure.

3.4. Info Structure

This section provides a suggestion for constructing the info structure, when used with SealBase() and OpenBase(). Note that the use of the aad and the info structures for these two functions is optional. Profiles of this specification may require their use and may define different info structure.

This specification re-uses the context information structure defined in [RFC8152] as a foundation for the info structure. This payload becomes the content of the info parameter for the HPKE functions, when utilized. For better readability of this specification the COSE_KDF_Context structure is repeated in Figure 2.
PartyInfo = {
    identity : bstr / nil,
    nonce : bstr / int / nil,
    other : bstr / nil
}

COSE_KDF_Context = [
    AlgorithmID : int / tstr,
    PartyUInfo : [ PartyInfo ],
    PartyVInfo : [ PartyInfo ],
    SuppPubInfo : [
        keyDataLength : uint,
        protected : empty_or_serialized_map,
        ? other : bstr
    ],
    ? SuppPrivInfo : bstr
]

Figure 2: COSE_KDF_Context Data Structure for info parameter

The fields in Figure 2 are populated as follows:

* PartyUInfo.identity corresponds to the kid found in the
  COSE_Sign_Tagged or COSE_Sign1_Tagged structure (when a digital
  signature is used). When utilizing a MAC, then the kid is found
  in the COSE_Mac_Tagged or COSE_Mac0_Tagged structure.

* PartyVInfo.identity corresponds to the kid used for the respective
  recipient from the inner-most recipients array.

* The value in the AlgorithmID field corresponds to the alg
  parameter in the unprotected header structure of the recipient
  structure.

* keyDataLength is set to the number of bits of the desired output
  value.

* protected refers to the protected structure of the inner-most
  array.

4. Example

An example of the COSE_Encrypt structure using the HPKE scheme is
shown in Figure 3. Line breaks and comments have been inserted for
better readability. It uses the following algorithm combination:

* AES-GCM-128 for encryption of detached ciphertext in layer 0.
* AES-GCM-128 for encryption of the CEK in layer 1 as well as ECDH with NIST P-256 and HKDF-SHA256 as a Key Encapsulation Mechanism (KEM).

The algorithm selection is based on the registry of the values offered by the alg parameters.

96_0({
  / protected header with alg=AES-GCM-128 /
  h’a10101’,
  / unprotected header with nonce /
  (5: h’938b528516193cc7123ff037809f4c2a’),
  / detached ciphertext /
  null,
  / recipient structure /
  [
    / protected field with alg for HPKE /
    h’a1013863’,
    / unprotected header /
    {
      / ephemeral public key with x / y coordinate /
      -1: h’a401022001215820a596f2ca8d159c04942308ca90
      cfbfca65b108ca127df8fe191a063d00d7c5172258
      20aef47a45d6d6c572e7bd1b9f3e69b50ad3875c68
      f6da0c9aa90c675df4162c39’,
      / kid for recipient static ECDH public key /
      4: h’6b69642d32’,
    },
    / encrypted CEK /
    h’9aba6fa44e9b2ceef9d646614dcda670dbdb31a3b9d37c7a
    65b099a8152533062’,
  ],
})

Figure 3: COSE_Encrypt Example for HPKE

Note that the COSE_Sign1 wrapper outside the COSE_Encrypt structure is not shown in the example above.

5. Security Considerations

This specification is based on HPKE and the security considerations of HPKE [I-D.irtf-cfrg-hpke] are therefore applicable also to this specification.

HPKE assumes the sender is in possession of the public key of the recipient and HPKE COSE makes the same assumptions. Some form of public key distribution mechanism is assumed to exist.
Since the CEK is randomly generated it must be ensured that the guidelines for random number generations are followed, see [RFC8937].

The COSE_Encrypt structure must be authenticated using COSE constructs like COSE_Sign, or COSE_Sign1.

6. IANA Considerations

This document requests IANA to add new values to the COSE Algorithms registry defined in [RFC8152] (in the Standards Action With Expert Review category):

6.1. HPKE/P-256+HKDF-256 and AES-128-GCM

* Name: HPKE_P256_HKDF256_AES128_GCM
* Value: TBD1
* Description: HPKE/P-256+HKDF-256 and AES-128-GCM
* Capabilities: [kty]
* Change Controller: IESG
* Reference: [[TBD: This RFC]]
* Recommended: Yes

6.2. HPKE/P-512+HKDF-512 and AES-256-GCM

* Name: HPKE_P521_HKDF512_AES256_GCM
* Value: TBD2
* Description: HPKE/P-512+HKDF-512 and AES-256-GCM
* Capabilities: [kty]
* Change Controller: IESG
* Reference: [[TBD: This RFC]]
* Recommended: Yes

TBD: More values to be added.

7. References
7.1. Normative References

[I-D.irtf-cfrg-hpke]


7.2. Informative References


Appendix A. Acknowledgements

We would like to thank Goeran Selander, John Mattsson and Ilari Liusvaara for their review feedback.

Authors’ Addresses

Hannes Tschofenig
Arm Limited
Email: hannes.tschofenig@arm.com

Russ Housley
Vigil Security, LLC
Email: housley@vigilsec.com

Tschofenig, et al. Expires 8 September 2022
Abstract

This specification defines how to represent cryptographic keys for the pairing-friendly elliptic curves known as Barreto-Lynn-Scott (BLS), for use with the key representation formats of JSON Web Key (JWK) and COSE (COSE_Key).

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/tplooker/draft-looker-cose-bls-key-representations.

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 2 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

This specification defines how to represent cryptographic keys for the pairing-friendly elliptic curves known as Barreto-Lynn-Scott [BLS], for use with the key representation formats of JSON Web Key (JWK) and COSE_Key. This specification registers the elliptic curves in appropriate IANA JOSE and COSE registries.

There are a variety of applications for pairing based cryptography including schemes already published as RFCs, such as Identity-Based Cryptography [RFC5091] Sakai-Kasahara Key Encryption (SAKKE) [RFC6508], and Identity-Based Authenticated Key Exchange (IBAKE) [RFC6539]. SAKKE is applied to Multimedia Internet KEYing (MIKEY) [RFC6509].
This branch of cryptography has also been used to develop privacy-preserving cryptographic hardware attestations schemes, including the Elliptic Curve Direct Anonymous Attestation (ECDAA) in the Trusted Platform Modules [TPM] specified by the Trusted Computing Group. Further work on similar schemes has also occurred at the FIDO Alliance [ECDAA]. Similarly, Intel released [EPID] which provides a solution to remote hardware attestation for Intel Software Guard Extension (SGX) enabled environments.

More recently, applications of pairing based cryptography using the Barreto-Lynn-Scott curves include the standardization effort for BLS Signatures [id.draft.bls-signature-04], which are used extensively in multiple blockchain projects due to their unique signature aggregation properties, including [Ethereum] [DFINITY] [Algorand]. Additionally, efforts are under way to standardize the general purpose short group signature scheme of BBS Signatures [BBS], which features novel properties such as multi-message signing and selective disclosure alongside zero knowledge proving. It is intended that this draft will help with these efforts by standardizing the associated cryptographic key representation in the popular formats of JWK and COSE.Key.

Other relevant work to this draft includes [JWP] which is extending the JOSE family of specifications to provide support for representing a variety of new proof based cryptographic schemes such as [BBS] which as referred to above uses the Barreto-Lynn-Scott curves.

There are multiple different pairing-friendly curves in active use; however, this draft focuses on a definition for the Barreto-Lynn-Scott curves due to them being the most "widely used" and "efficient" whilst achieving 128-bit and 256-bit security (BLS12-381 and BLS48-581 respectively).

More extensive discussion on the broader application of pairing based cryptography and the assessment of various elliptic curves (including the BLS family) can be found in [id.draft.pairing-friendly-curves-10].

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.
2.1. Representation Definition

The following definitions apply to the pairing-friendly elliptic curves known as the Barreto-Lynn-Scott (BLS) curves.

2.1.1. JSON Web Key Representation

When expressing a cryptographic key for these curves in JSON Web Key (JWK) form, the following rules apply:

* The parameter "kty" MUST be present and set to "OKP".

* The parameter "crv" MUST be present and value MUST be one defined in Section 2.1.3.

* The parameter "x" MUST be present whose value represents the curve point for the public key. This value MUST be encoded using the serialization defined in [id.draft.pairing-friendly-curves-10] Appendix C and MUST be base64url encoded without padding as defined in [RFC7515] Appendix C.

* The parameter "d" MUST be present for private key representations whose value MUST contain the little-endian representation of the private key base64url encoded without padding as defined in [RFC7515] Appendix C. This parameter MUST NOT be present for public keys.

2.1.2. COSE Key Representation

When expressing a cryptographic key for these curves in COSE Key form, the following rules apply:

* The parameter "kty" (1) MUST be present and set to "OKP" (1).

* The parameter "crv" (-1) MUST be present and value MUST be one defined in Section 2.1.3.

* The parameter "x" (-2) MUST be present whose value represents the curve point for the public key. This value MUST be encoded using the serialization defined in [id.draft.pairing-friendly-curves-10] Appendix C.

* The parameter "d" (-4) MUST be present for private key representations whose value MUST contain the little-endian representation of the private key. This parameter MUST NOT be present for public keys.
2.1.3. Curve Parameter Registration

<table>
<thead>
<tr>
<th>JWK &quot;crv&quot; value</th>
<th>COSE_ &quot;crv&quot; value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bls12381G1</td>
<td>TBD (13 requested)</td>
<td>A cryptographic key on the Barreto-Lynn-Scott (BLS) curve featuring an embedding degree 12 with 381-bit p in the subgroup of G1 defined as ( E(\text{GF}(p)) ) of order ( r )</td>
</tr>
<tr>
<td>Bls12381G2</td>
<td>TBD (14 requested)</td>
<td>A cryptographic key on the Barreto-Lynn-Scott (BLS) curve featuring an embedding degree 12 with 381-bit p in the subgroup of G1 defined as ( E(\text{GF}(p^2)) ) of order ( r )</td>
</tr>
<tr>
<td>Bls48581G1</td>
<td>TBD (15 requested)</td>
<td>A cryptographic key on the Barreto-Lynn-Scott (BLS) curve featuring an embedding degree 48 with 581-bit p in the subgroup of G1 defined as ( E(\text{GF}(p)) ) of order ( r )</td>
</tr>
<tr>
<td>Bls48581G2</td>
<td>TBD (16 requested)</td>
<td>A cryptographic key on the Barreto-Lynn-Scott (BLS) curve featuring an embedding degree 48 with 581-bit p in the subgroup of G1 defined as ( E(\text{GF}(p^8)) ) of order ( r )</td>
</tr>
</tbody>
</table>

Table 1

3. Security Considerations

See [id.draft.pairing-friendly-curves-10] for additional details on security considerations for the curves used. Implementers should also consider the general guidance provided in Section 9 of [RFC7517] and Section 17 of [RFC8152] when using this specification.

Furthermore, because this specification only defines the cryptographic key representations and not the usage of these keys with specific algorithms, implementers should be aware to follow any guidance that may be provided around appropriate usage of the keys and or additional steps that may be required to validate the keys within the context of particular algorithms.
4. IANA Considerations

4.1. JSON Web Key (JWK) Elliptic Curve Registrations

This section registers the following values in the IANA "JSON Web Key
Elliptic Curve" registry [IANA.JOSE.Curves].

Bls12381G1
* Curve Name: Bls12381G1
* Curve Description: 381 bit with an embedding degree of 12 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  \( E(GF(p)) \)
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.1

Bls12381G2
* Curve Name: Bls12381G2
* Curve Description: 381 bit with an embedding degree of 12 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  \( E'(GF(p^2)) \)
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.1

Bls48581G1
* Curve Name: Bls48581G1
* Curve Description: 581 bit with an embedding degree of 48 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  \( E(GF(p)) \)
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.1
Bls48581G2
* Curve Name: Bls48581G2
* Curve Description: 581 bit with an embedding degree of 48 Barreto-Lynn-Scott pairing-friendly curve using the r-order subgroup of \(E'(\text{GF}(p^8))\)
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.1

4.2. COSE Elliptic Curve Registrations

This section registers the following value in the IANA "COSE Elliptic Curves" registry [IANA.COSE.Curves].

Bls12381G1
* Curve Name: Bls12381G1
* Value: TBD (13 requested)
* Key Type: OKP
* Curve Description: 381 bit with an embedding degree of 12 Barreto-Lynn-Scott pairing-friendly curve using the r-order subgroup of \(E(\text{GF}(p))\)
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.2
* Recommended: Yes

Bls12381G2
* Curve Name: Bls12381G2
* Value: TBD (14 requested)
* Key Type: OKP
* Curve Description: 381 bit with an embedding degree of 12 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  $E'(GF(p^2))$
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.2
* Recommended: Yes

Bls48581G1
* Curve Name: Bls48581G1
* Value: TBD (15 requested)
* Key Type: OKP
* Curve Description: 581 bit with an embedding degree of 48 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  $E(GF(p))$
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2.1.2
* Recommended: Yes

Bls48581G2
* Curve Name: Bls48581G2
* Value: TBD (16 requested)
* Key Type: OKP
* Curve Description: 581 bit with an embedding degree of 48 Barreto-
  Lynn-Scott pairing-friendly curve using the r-order subgroup of
  $E'(GF(p^8))$
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
5. Normative References


6. Informative References
Appendix A. Acknowledgments

The authors would like to acknowledge the work of Kyle Den Hartog, which was used as the foundation for this draft.
Appendix B. Document History

-00

* Initial version

Authors’ Addresses

Tobias Looker
Mattr
Email: tobias.looker@mattr.global

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: https://self-issued.info/
CBOR Web Token (CWT) Claims in COSE Headers
draft-looker-cose-cwt-claims-in-headers-00

Abstract

This document describes how to include CBOR Web Token (CWT) claims in the header parameters of any COSE structure. This functionality helps to facilitate applications that wish to make use of CBOR Web Token (CWT) claims in encrypted COSE structures and/or COSE structures featuring detached signatures, while having some of those claims be available before decryption and/or without inspecting the detached payload.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/tplooker/draft-looker-cose-cwt-claims-in-headers.

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 2 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

In some applications of COSE, it is useful to have a standard representation of CWT claims [RFC8392] available in the header parameters. These include encrypted COSE structures, which may or may not be an encrypted CWT and/or those featuring a detached signature.

Section 5.3 of the JWT RFC [RFC7519] defined a similar mechanism for expressing selected JWT based claims as JOSE header parameters. This JWT feature was motivated by the desire to have certain claims, such as Key ID values, be visible to software processing the JWT, even though the JWT is encrypted. No corresponding feature was standardized for CWTs, which was an omission that this specification corrects.

Directly including CWT claim values as COSE header parameter values would not work, since there are conflicts between the numeric header parameter assignments and the numeric CWT claim assignments. Instead, this specification defines a single header parameter registered in the IANA "COSE Header Parameters" registry that creates a location to store CWT claims in a COSE header parameter.
2. Terminology

3. Representation

This document defines the following COSE header parameter:

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Value Type</th>
<th>Value Registry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwt</td>
<td>TBD (requested allocation 11)</td>
<td>map</td>
<td>[IANA.CWT]</td>
<td>location for CWT claims in COSE headers</td>
</tr>
</tbody>
</table>

Table 1

4. Privacy Considerations

Some of the registered CWT claims may contain privacy-sensitive information. Therefore care must be taken when expressing CWT claims in COSE headers.

5. Security Considerations

In cases where CWT claims are both present in the payload and the header, an application receiving such a structure MUST verify that their values are identical, unless the application defines other specific processing rules for these claims.

6. IANA Considerations

IANA is requested to register the new COSE Header parameter in the table in Section 3 in the "COSE Header Parameters" registry [IANA.COSE].

7. Normative References


8. Informative References
Appendix A. Document History

-00

* Initial version

Authors’ Addresses

Tobias Looker
Mattr
Email: tobias.looker@mattr.global

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: https://self-issued.info/
Abstract

This document describes JSON and CBOR serializations for several post quantum cryptography (PQC) based suites.

This document does not define any new cryptography, only serializations of existing cryptographic systems.

This document registers key types for JOSE and COSE, specifically PQK, CRYDI, pset.

This document registers signature algorithms types for JOSE and COSE, specifically CRYDI3 and others as required for various post quantum signature schemes.

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 6 September 2022.
Table of Contents

1. Notational Conventions ........................................ 3
2. Terminology .................................................... 3
3. CRYSTALS-Dilithium ............................................. 3
  3.1. Overview .................................................. 3
  3.2. Parameters ............................................... 5
    3.2.1. Parameter sets ...................................... 5
  3.3. Core Operations ........................................... 5
    3.3.1. Generate ............................................ 5
    3.3.2. Sign ................................................ 5
    3.3.3. Verify .............................................. 5
  3.4. Using CRYDI with JOSE ...................................... 6
    3.4.1. CRYDI Key Representations ........................... 6
    3.4.2. CRYDI Algorithms .................................... 7
    3.4.3. CRYDI Signature Representation ..................... 16
  3.5. Using CRYDI with COSE ..................................... 20
4. Falcon ......................................................... 20
5. SPHINCS+ ....................................................... 20
  5.1. Overview .................................................. 20
  5.2. Parameters ............................................... 22
    5.2.1. Parameter sets ...................................... 22
  5.3. Core Operations ........................................... 22
    5.3.1. Generate ............................................ 22
    5.3.2. Sign ................................................ 22
    5.3.3. Verify .............................................. 22
  5.4. Using SPHINCS+ with JOSE .................................. 22
    5.4.1. SPHINCS+ Key Representations ....................... 22
    5.4.2. SPHINCS+ Algorithms ................................ 22
    5.4.3. SPHINCS+ Signature Representation ................. 23
6. Security Considerations ........................................ 23
  6.1. Validating public keys .................................... 23
  6.2. Side channel attacks ...................................... 23
  6.3. Randomness considerations ................................ 23
1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Terminology

The following terminology is used throughout this document:

PK  The public key for the signature scheme.
SK  The secret key for the signature scheme.
signature  The digital signature output.
message  The input to be signed by the signature scheme.
sha256  The SHA-256 hash function defined in [RFC6234].
shake256  The SHAKE256 hash function defined in [RFC8702].

3. CRYSTALS-Dilithium

3.1. Overview

This section of the document describes the lattice signature scheme CRYSTALS-Dilithium (CRYDI). The scheme is based on "Fiat-Shamir with Aborts"[Lyu09, Lyu12] utilizing a matrix of polynomials for key material, and a vector of polynomials for signatures. The parameter set is strategically chosen such that the signing algorithm is large enough to maintain zero-knowledge properties but small enough to prevent forgery of signatures. An example implementation and test vectors are provided.
CRYSTALS-Dilithium is a Post Quantum approach to digital signatures that is an algorithmic approach that seeks to ensure key pair and signing properties that is a strong implementation meeting Existential Unforgeability under Chosen Message Attack (EUF-CMA) properties, while ensuring that the security levels reached meet security needs for resistance to both classical and quantum attacks. The algorithm itself is based on hard problems over module lattices, specifically Ring Learning with Errors (Ring-LWE). For all security levels the only operations required are variants of Keccak and number theoretic transforms (NTT) for the ring \( \mathbb{Z}_q[x]/(x^{256}+1) \). This ensures that to increase or decrease the security level involves only the change of parameters rather than re-implementation of a related algorithm.

While based on Ring-LWE, CRYSTALS-Dilithium has less algebraic structure than direct Ring-LWE implementations and more closely resembles the unstructured lattices used in Learning with Errors (LWE). This brings a theoretical protection against future algebraic attacks on Ring-LWE that may be developed.

CRYSTALS-Dilithium brings several advantages over other approaches to signature suites:

* Post Quantum in nature - use of lattices and other approaches that should remain hard problems even when under attack utilizing quantum approaches
* Simple implementation while maintaining security - a danger in many possible approaches to cryptography is that it may be possible inadvertently introduce errors in code that lead to weakness or decreases in security level
* Signature and Public Key Size - compared to other post quantum approaches a reasonable key size has been achieved that also preserves desired security properties
* Conservative parameter space - parameterization is utilized for the purposes of defining the sizes of matrices in use, and thereby the number of polynomials described by the key material.
* Parameter set adjustment for greater security - increasing this matrix size increases the number of polynomials, and thereby the security level
* Performance and optimization - the approach makes use of well known transforms that can be highly optimized, especially with use of hardware optimizations without being so large that it cannot be deployed in embedded or IoT environments without some degree of optimization.

The primary known disadvantage to CRYSTALS-Dilithium is the size of keys and signatures, especially as compared to classical approaches for digital signing.
3.2. Parameters

Unlike certain other approaches such as Ed25519 that have a large set of parameters, CRYSTALS-Dilithium uses distinct numbers of parameters to increase or decrease the security level according to the required level for a particular scenario. Under DILITHIUM-Crustals, the key parameter specification determines the size of the matrix and thereby the number of polynomials that describe the lattice. For use according to this specification we do not recommend a parameter set of less than 3, which should be sufficient to maintain 128 bits of security for all known classical and quantum attacks. Under a parameter set at NIST level 3, a 6x5 matrix is utilized that thereby consists of 30 polynomials.

3.2.1. Parameter sets

Parameter sets are identified by the corresponding NIST level per the table below:

<table>
<thead>
<tr>
<th>NIST Level</th>
<th>Matrix Size</th>
<th>Memory in bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4x4</td>
<td>97.8</td>
</tr>
<tr>
<td>3</td>
<td>6x5</td>
<td>138.7</td>
</tr>
<tr>
<td>5</td>
<td>8x7</td>
<td>187.4</td>
</tr>
</tbody>
</table>

Table 1

3.3. Core Operations

This section defines core operations used by the signature scheme, as proposed in [CRYSTALS-Dilithium].

3.3.1. Generate

See [CRYSTALS-Dilithium]

3.3.2. Sign

See [CRYSTALS-Dilithium]

3.3.3. Verify

See [CRYSTALS-Dilithium]
3.4. Using CRYDI with JOSE

Basing off of this (https://datatracker.ietf.org/doc/html/rfc8812#section-3)

3.4.1. CRYDI Key Representations

A new key type (kty) value "PQK" (Post Quantum Key Pair) is defined for public key algorithms that use base 64 encoded strings of the underlying binary materia as private and public keys and that support cryptographic sponge functions. It has the following parameters:

* The parameter "kty" MUST be "PQK".
* The parameter "alg" MUST be specified, and its value MUST be one of the values specified in table "TBD".
* The parameter "pset" MUST be specified to indicate the not only parameter set in use for the algorithm, but SHOULD also reflect the targeted NIST level for the algorithm in combination with the specified parameter set. For "alg" "CRYDI" one of the described parameter sets "2", "3", or "5" MUST be specified. Parameter set "3" or above SHOULD be used with "CRYDI" for any situation requiring at least 128bits of security against both quantum and classical attacks.
* The parameter "x" MUST be present and contain the public key encoded using the base64url [RFC4648] encoding.
* The parameter "xs" MAY be present and contain the shake256 of the public key encoded using the base64url [RFC4648] encoding.
* The parameter "d" MUST be present for private keys and contain the private key encoded using the base64url encoding. This parameter MUST NOT be present for public keys.
* The parameter "ds" MAY be present for private keys and contain the shake256 of the private key encoded using the base64url encoding. This parameter MUST NOT be present for public keys.

Sizes of various key and signature material is as follows (for "pset" value "2")
<table>
<thead>
<tr>
<th>Variable</th>
<th>Paramter Name</th>
<th>Paramter Set</th>
<th>Size</th>
<th>base64url encoded size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>sig</td>
<td>2</td>
<td>3293</td>
<td>4393</td>
</tr>
<tr>
<td>Public Key</td>
<td>x</td>
<td>2</td>
<td>1952</td>
<td>2605</td>
</tr>
<tr>
<td>Private Key</td>
<td>d</td>
<td>2</td>
<td>4000</td>
<td>5337</td>
</tr>
</tbody>
</table>

Table 2

When calculating JWK Thumbprints [RFC7638], the four public key fields are included in the hash input in lexicographic order: "kty", "pset", and "x".

3.4.2. CRYDI Algorithms

In order to reduce the complexity of the key representation and signature representations we register a unique algorithm name per pset. This allows us to omit registering the pset term, and reduced the likelihood that it will be misused. These alg values are used in both key representations and signatures.

<table>
<thead>
<tr>
<th>kty</th>
<th>alg</th>
<th>Paramter Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQK</td>
<td>CRYDI5</td>
<td>5</td>
</tr>
<tr>
<td>PQK</td>
<td>CRYDI3</td>
<td>3</td>
</tr>
<tr>
<td>PQK</td>
<td>CRYDI2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3

3.4.2.1. Public Key

Per section 5.1 of [CRYSTALS-Dilithium]:

The public key, containing p and t1, is stored as the concatenation of the bit-packed representations of p and t1 in this order. Therefore, it has a size of 32 + 288 kbytes.
The public key is represented as x and encoded using base64url encoding as described in [RFC7517].

Example public key using only required fields:
Example public key including optional fields:
{
  "kid": "key-0",
  "kty": "PQK",
  "alg": "CRYDI3",
  "key_ops": ["verify"],
  "xs": "z3u2QVjflnRZDSZnle8g4oKH4YUU6TnvpuKU4WrrGdXw==",
  "ds": "5Bu28Xo4jQirc/5E237BcoGo+Ho/TTjI+GULLTcIySU==",
  "x": "z7u7GwshsijnfHH3Nkr2s2xvvw020Rcw5ymdlTnhRnJddOrOon+nfXRVUZVy9qIfz5n7yebt7KjmrbN1nu2/zaFKLluizMcLNp6LW0WbMqxoC2XOVNAW3QjrCcGU23Nn/indmtmWz5NPr43E592Sctt5M+SVlfQeYv8pHmtkQknE8/j/7Tr7gNpviU7vnXmhWHTMJ4i/zz0Gxgq43odFtb0edKNT/enyu+VvUgoiJ6c8C/1B6o1W1YHeaL0BIEFFBAIAh/vn/cUyMavPqsJuTejete32kGCDD7jumew2t6O6D1lhB/2ZJXm8mcbxFi6krmrXixQXj9j/9Vln1fEzd3VFpces/3ComsFGqmTpLDK+AvW+SWVkd2NKq7hL/Ayx1w2u2cqrErq2UTS/24ic6v8kZfzr3gRmHuDKuF5t8j1e2/yVvqgDjpW02egCKE12G8d8uhdUedeCr58pi10l/UXy5awgCcZTdeCjn1OPObGoots9719gw1x44vZCQUKKVD5PZu2glGQDuUXXS01cNTcMms/miEFmmOzb88jxULpblv19HoQ3ocMo2Ou4AZRt9G/L07MwcuioUFCTwAfau+2qgNAzn/A51010j2N0LltAaoXofo/Ctznscrt02MYGmmoQ99dahkPvuFqOC08hDtLplnpq31I1llrFOQ/nJcn9SvKhU87COBl6g8v4cW5b9eU2xqv2x2fVDy9l0m+5S+M6jhtSLyLypKJXQTt3/zi3797vIw5n9vHpA+PliUUw9tgg6F6x8WEGSN1O0n1xntM3gie9vEjXyp5s/kBEFZI/DCyFzstaIrr2GadNhae+PlJFZzJwN5jaLwzlqzw3F3yTzh0queSGBA+FgKixCqgn2wn1/vz0Dkhr6cMaUooLe0FscU1nA21Z3W6R2Tn2EusyRs/ht2RZjxxFeC213nLeEDFCC/vKNDnx QBFSFzpZJkkGtJEXEs3mnp+CbBVMBrbagsLnzsAGLYbnrovVmATU51q6LGyBpoF+N/Rkzq7Zx6C2PukMCQoboGuW6N6u6cMNhr5sAyGk1ZBiW82c7N0u0s2lpCcqNGwmt1tLw/8969Kyo6cS7YXAZa40QcvoYSwnxxQBPBTRJ+wBoSvpGbltLDy9w3pXeXN5nnao340D2sA/7YEM1IqcTHCAv3F9B9w670Cf7Qml6gbd6vdmUovDE+e0er7IAnWMRgviIZvY9sKEQrcMua/2q5L5xbSbD05KRf8ZAAZ2B8iSCDrlnhrXQXZBKBKjsiCvVqUXzrwrGEQqgRmbk4f5vGYCG/4l/O8Knorur+jfj6oWQYKFyq1uQGRL1XHKUGUG1xfv03r7UbJugycjV0s5dbGxho2skqogB3/ZEpkfavrNoxcetv/vq4pJ8J1Y97GDbO0MGvHmbdHugjMTgVHJFZBpBPimmr+rem70+4gOr7ywRxcVtv1zIlwtpPBwaf/IXDn5N5esexg1rlyrCRNmB08b0O1sxLwmxtC5e4F90/lUKaas7qj3A3AQ70O86Dqd1pQkqshsFBy+wJSQvuOCYMMCp6IMnJxND321ChnLiran6Ve/F2507ST0YwNXdnesMrZTmHuKHsAh0UK9WABTcMc5n6wG601aONsFrldg+KmHXN6hF/onwbb/EzD155TAll8wRqrj1h2xv66e9XRKerpaOW/Fyk+2mWbBP/98VE12/EwOGqAIlP/eIaVeEm0lRkp0l6gBjmsyYtHnuWgXcv5Qiy7/eGw9Zjvb3J3G3jxvbvExqDyfDo067EK1/5wDFXPYU0JfjKpekXvqzUrQs94B9tcRvMt5ndEVE21TFFFz/RBB7Rxn2ZLIh+hQc+c9c+PeeAedwGwoa0m1Ic/c1y/61ymodO6Hxsfw1MW7gej+iHn4InaeeEgh1019kMGTZblq5pC8PZ/6FLzKTBMJWq90/Oblv1lM9EPblcLeF/bRIA2ZUD62Fid2Tn6Epn3QvqegDq4Pm1EBT/GwY9v2M5/0Bd6d6d1Hq_PWbKxH4x56TmowFkhM2imn0yVupwxoRJrU1HRMBC3tn8C4/2pFl+sGv3Gl3ptkISLPKQkTQ16DmWxExdbvdty1s2H2acpc1LDiA/yEyT44R2m3VJN9JR/6Hx3teqNleqRsz/xMIKCDwWr1Czy/-6y1Q5owybR9cjIty/LQX2gtaARl5XON99U0BVv+/Z/E03QVQ+Ecsc781S8B/6n6FCzl1bk/HAfA+cu0yMoBmGeEM8W3mTuSp54JBAcWk5w0XWNNQ/DSWVEdgzULGqPh+yHEXvzj3vRfELhkJ8H97y2A+7RXXXUZH/JoNOGHUqUpUG/bF0k2tnaIHC/xsXMUbdCr3CfftFFHgk1svtcEDEFDxk1iA8pGsa5j9T0do6n3WelnBDu1oskNfmpcVke/U13XpovuoW3BGwBUcWpPs47D7RgRkx11bSaEYLY1hV2aVshcvzg46Akoq+Q7TjckDF/8u8sSQk0AbuhxWFPQsSiB8OZ/Ur=

3.4.2.2. Private Key

Per section 5.1 of [CRYSTALS-Dilithium]:

The secret key contains $p,K, tr, s_1, s_2$ and $t_0$ and is also stored as a bit-packed representation of these quantities in the given order. Consequently, a secret key requires $64 + 48 + 32(k+l) \ast dlog (2n+ 1)e + 14k$ bytes. For the weak, medium and high security level this is equal to $112 + 576k + 128l$ bytes. With the very high security parameters one needs $112 + 544k + 96l = 3856$ bytes.

The private key is represented as $d$ and encoded using base64url encoding as described in [RFC7517].

Example private key using only required fields:

```
{ "kty": "PQK",
  "alg": "CRYDI3",
  "x": "z7u7GwshsjnfhHH3Nrs2xvvw020Rcw5ymdlTnhRenjDdrO0+nfXRVUVZVYy9q1l/5zDn77zTgrIsk3M3WX8bqs1c+B1fg121a/AxWd2jcd66+yJKctkGH2O9R7vycOC221MznW/g17yebl7JkmjrBnIN/u2fAKFLuiziMcLNP6LVmWbMxqoC2X0VQNAWX3qjXrcCggU23Nr/imtdmWz5Npr4E5S92Sct57m5+SVlgQeYyVhMhtqKqenE8/jr7TgrNpuiV7nXhHWTM4jIzoGxq43odFFthboEdKNT/enyu+VvUOgiJ6cn8C/1B6o1W1YHeaL0BEIFFbAiaH2/vnf/cUYMaVPqsDjuETsjetcE32kGCD7Jkume2t06D11hB/ZZZ2X9mckbxEF6KrmXIrXQj9\9L1n1fEzd35Fvpc/C3omsFgqmntPdpL+AVW/SWVkd12NKq7hL/Ayx1W2u2cqVErQZUTS/\+rhc6V8kZ3frxgRmH0kUF5Btje2/yVvqqPjwPOzegCFK32Gd8duhcUde7CR5s110/Ux5y5AwCgZTDVcJnIOPobGoots91t19w1x4vnCZQVUKDVZu1gIKGqDUYX80lCnTJCMs/miFEmmnOzvB88juXLPb1v93xHorQ3o2mz0u4AZRt9g/L07Mwcui0FCWtaIAu+2qgNaA/n/A510l1jN0LttA0oxof+Tczsccr02MYGhmoQ9daHkpVugq0C08dtLPnq3lQ1IfROQ/jcN5vNBKu87COBju52D+iL8V4zy8FNO59MCsb9UCLw2xvfdIIjs9/j7ThTQcvrX8VpX-md42yFQFswG5NaloefmN8W49E5dmevc8AJAtwDirRBDVf9p3x+5S+Mjht0eSLv/YpKJQXCTI/za1379vICHwkn9vBPa+P1UWu9tgF6xF8xWEGSN1o0v1nxMt3g1vehjYxJsp/5KBEFZI/DCyFzstAirJ2GadNhae/P1JFZwWnX5jaLwzldqzu2F3yTzh04sgBAA/kfXgcn2nwl/vz0Dkbmx6MaC0o0Fscu1nAa1329W64Lt2nEuY0ntRhx2ZjxxFc2lx3neLeDFEC/NEKNQDQFBSFzp3JKKgtJtxE23p+CbBVMrbagsLznasAGLYbnrocVmATU1qrf6LyBpufs+N/Rkq7ZXh6CZPuMcGQbcOuW60BuuMNh1r5ayGk1zBiW82C7Nu0hs2pLcgmNqYm1i+LW/8R96kypo5c784YAZA40QvqovySmwqXFBTRJR+wB0SvPqBLTxdY9Gw3pXeXN5n304d2ZAZ/7YEM1qCTHCav3F89ew17Fq1mg6bbuuoVdVY+poer7IAmWMRgviizYy9eKEEQmua/2qL5xFsBdD05Krf8AZA2BB1SCDmFr1nZxRQzXbBKJivsCQDuxwreGEQQqRmbk4f5GyCG/4i/08Knour+jjF6wWVQD1Kfyq1QQU1XHKUG1Xf03r7UbJugycjV0skGxho2kgq0qgz/ZEpkefvrNoxeotw/z4qqjI8J1Y19G7Db0mGVHbmdHugMVTghVJFJJpImnnt+1m7b0/4qOr7ywRxCVt21ziWtPBPWaf/1DnX5Ngesex1gR1YrcTRNmb80b01sxLQmxcT4e0Q/0
```
LUkas7qTJ3AQThOfDttIpqtshsBFy+WjsQuoXCMYRcpi6mlpxJndDF321bCnllranV6e
F2ST05YT+NWNSDesMzTRmNhUHW5Ahu0k59WAVbcM5ba0q6l01aNsFrcLag+KhnX6HNPn
oobwJtEsdi5ST7a1W8rjqii28x6h9eRRKepaOw+Fy+2MwPByp/98V1E2+WqoAilNp
elAvUeoM0lkriGFb46bJymiYUhWngcv5Qiy7/ewG92pvB3J3G3jvxvbyenExqdYFdc067Eki
5WdfDFzUjKjKpeKvNZvThQrgs5Bc6RytM5ndEVE21TPFP1/RBB7Rnxn2L+Kq+c+c
pEEwAeW60AI1Mlcp/1CyY16mdO6RRhnxwfl1M7gej+hN41kaosghI19hkMTGLzSBq5sp82
6F2LKTBMJw9g/0bIvi1bmi6g6Rb1pc1ef/Br1A2UTD62fd2T6X6Enp30Vqgeq/PfMlEBF
Gw91v2m6/08D621HfKqKBDK9F56QowaKFMH2iminz0YVupworoJrulRHMCb3tn8C4
ZpFl+sAVG3Vij3tK157PKQKTPIdI6MwxBzbrdvyT1sh2aqgpc1LdLsA/yFT4RR32vMNJR9
6N3txeqN1e6R6XMD/MLcdWr1c1j/6yeIYWbrc9Cjity/tLQX2gtARlS0X9h9UBVv+z
E03V0z+Ecsc781BG96/n6Fz1y3bK/HaGF+cuoyMbGmEN8m3WmTUp5sJBAcKw5x0WQNNG
DwV6dwgzuLHgpo+hYExdjoVz2eL1hKh8yq2A+7RXXUXH1/3j0NNOGnupHUGo/Fbo3ktaILC
xX0XMUDC3CviteF8RHSk1vscERDFxK1hAR8Ga59j70d06m3ewBNbDU1soaKNPUpmcVkc
UL13XpxuoW3bgCwJzBUCWcvps47DJRgGx011sbAeYY1hTVaaShcvzg4aAcko+Q7jgdkcf
/8uszSQQ0AbuhxWQPQs1B8PB07/U="}

"d": "Z7u7GwhsijfnH3krsz2xvxxV0w20C6w5ymlTnrenjDBuNgL6F1KuLRu52b5MT5
yr15F9oDkw+W2sUrSvMfd7V7gEY897mUYF352QW0Qm92X0I0CH+GGF0K56b5FOF6x9V8
UDqnopY2JR1ROhdijucUnXa1YiA1XAbUBoEFOUHAHGBzQ4A2dxdYV0TQRV303gu4
GI1SeyE3IUMwD31IUIMdSIVehwBYtIyWGI3vmJVI1U2ERoHUBoJ2q8RFuTHy
BSqnhBIG1A0oMV2CMNtUXQINGKHzQumxQ3dEgBnhQyOgmFjxdyl1CjdGgBSEBI4j
CEJMBGJQWRN3jmrSNWQ1QJnjdcnM1hluJ31MrdIn4md4wH1IDEYcAhElBCQ
J1E5A1BWQzy1AIocBccZFCgVca25DNCVBtJgzaAznQGYH11WvzYXQrCkB1BZV4
VxZv1M4l1XgYIof1VHc1InE1hEh1Urjg90cxJ0CSCwVUVCcHzdUDTASiCAw
UIjAg1il0q1wpmjmcACzXczHvZV4h4EnnNAWVzObJNnNEcicTdyEeUHUBFvJETENnjd
YUFnFDPXNUHUVCFwJo2A11wFhYdC0zmdJ1i3uhhYYkAgyRv1izdv
BERVhJNQAFMWikUhZCEQXq1Thty/R1IFGGNNTkAcwdrNJBNgEYEd2RVEwUDMDzWAAhNQEEVMAs
hUSGtT3V1DX0LBFxPkhVCCZEBAjYjcACxGfnknHqK11YsHHNEBeSmqK9uZWRSVd
GDZzRCCQ1nHe3j1H3g5SFGY80YCVltHNCVHmxzA2TSAGMVQushkBIFRFQBVBc0HNGEEJTVcQSVQO
hZq1iBkZBYQ40R2MV1udwVCtKiQYeeFYEREYZ2NERyFvDHJzAdZEHBmhdSCFIgh1iWB
AxxNzFjEpOuWFCF4AhzJR3JSEOREETHxAGYBjTggcdUfDjBOFmncY3UUFCAy1AnZ3BVTs
ECCBQYBNEAdqZ3K9H1E3zlIFxQVCgQBjEFDcIV01AacFqio1AgAUVQAhEInQWECLtU14
E2UOMAKQskzsdK3i50hQin4hQdGQcEEdTChrhD3fMBBmmY2geRKNWdllAVKFTVEGC
ghzrRXyBYBggzKGVXJUhoHEYI1yqFmHgZwmgEwWWVvoIsd4vlh4d4VMUhgS2XHUXsACgmt3J3Eg
QMI3zI1DThiWRswAfr2QBQjQjE501hnB111B1WZC8hDMyfBM2Q0UJXcZnYF
RTUFVQqoXmujB0AhJXjHDzXBBECXATUEDq3BtAchwiNOMwCB31ugBEArxVURdxd
QqVhBBFEBB4cQgRhG4ugCrcydaUQBr0UjYjjeBյУhSBjJlyV2CXYbcElmC1iN4ED
dgUVGHCMyCHRCQ8I0Qd4hDFJ31jld2gyQyC1rJ3eWcXJN3bOHyOuo2dHAtArew
OBeBQoQjYchdQjMTRX6B3I3AEjEXxheEEmghIAAGdjudVbnd12FCAvCvyx4ciz2yZDEv
E1EhCXYIjaCQXCaercoIDRJWksCNRvhEFP3M3JGCQKpyYWykndRA1qTH0MFqzyYcCY
wciAFRIN31AVECBi3Mqz1ikUHrLcldI1NSAFXN0U0nnRcvQUSyn83h3Ny1MQWVEQV
E1EhOnQyKNb0Lc1CCdmmjYl1GHVU1A0CHMTZ1URbgR4ai1WjQj1DDMYhwNGNw1GcIB0dA
NXMBERTXyCI3jKMN1I1AABA4bEMUDuwhdFjKQEdERlM1dZehZm02hGAHYMgZ4nEHEnh
U1Z21vCBB2XU0mcrZBnuEBXEUWdQ4RHHUXZOEzQogABHRAISEVzedQhRmbqgWEY
Zzucmz4Qd2wqgjQQuMsCqgIIPQ3YFchzYObi11VCYGBAvmVAFd1hMnm21hnhwDTrRjJy
IzzCwgiF1FcaCH1U1WyueAFAW3VR92Xq2QYAgMYUjP7wmmWEPw7UKk131L1aly/6dN4n
dBriAYS8JnKvq6pBeF07cix95sRFA07Ex7VREYyVh9QR0QpELBmcfdfhcZCKWM0o
OBF7X1WMR9Q0q4ybtPdJKGQK12yCkUKJHR2+uwe182qyc8KFDmdOzIX08KaA98eUy6SR
pFpKD6D+xGtJ0FWtYnp1Ly2AI3G5JbH5OoShDv1vccGkF49gPVKwMqJQspqg7dAjiJ"
Example private key using optional fields:

```json
{
    "kid": "key-0",
    "kty": "PQK",
    "alg": "CRYDI3",
    "key_ops": ["sign"],
    "xs": "z3u2QVjfhnRZDS2nle8g4oKH4YUU6TnpvkU4WrrGdXw=",
    "ds": "5du28XoJqIr3c5TE2E3tGoH0+JT1j+UULLXtCiSy=",
    "x": "z7u7GwhsddfjnnH3nkr2sxv2v020rcw5ymdlTnhIREn+0nfvXRUVZVYy9q1l5zDn77zTgr1skM3XW8bqsclc+Brf212a/wxD2jci1d6j+yjKtckGH260R7vc03C221mZv\zgl7ypebT7jkmmrB2Nl+u+zAFEKfLuzIcMLnPF6WbMxqoC2XO0VNAW3QjrCcGUGU3Nc\intdmmw3NPR4F3E592Stct5M+Sv1f9qEy8pHmtdkknE8/jr7TgrnpuiV7nXwmHWTMJ4im\zoxGkgq43odFFthoEdKNT/enyu+VvUGoJ16cN8C/1B6o1W1hEaLOEJFFBAIAh2/vnf\cUYVmAPq5Su7tjcteE32kJGD7jku3e2068D1lhB/222JX8mckbF6KrnxIRXqg9j9\9Ln1vnEzdf3zvFpcs/C03sFmgTnpLDK+A+VWSVkd2Nkg7hL/Ayx+hT2u2cqqVfRQ2UTS\ZicxV68k2f3grMnHOKuF5tje2Z/yVvqgjPjwPO2eCgKE1J2D8d8duhUede7CR5rpi0\UXy5AwgCZzTeDexcZ0PbGoota919gw14vnZCQUKVP2zU1gKQ6gUXYS01cNTJM\c\miFEmmmOzVbB89xjuLPbLb19hQ0Q3ocM20z4A4ZT9g/L07Mwcu1u0FCWTA0a2+a2gqNA+n\ZAS1010j2N0LttAaOxoxF+Ctzscrt02YgHm0Q99aHk6vpUeq0c08hDtlplnq31Q1IIrFROQ\jcNs9Vn9U9k1Xhu9N9pA+P1Uuw9Tf6Xp8WEGNSL10On1vxtM3givhejYj5p5/bKEFZI\DCyFzstAir2J2adnHnae+P1JFZzJWnX5jaLwzldquwzF3ytZnho4qgBA+fkOxTcqn2nwl\vz0d0khx6rcMaouo10eFsccU1nAa1Z39W64LT2nEuvYsr/xht2RzJxxFc21X3nLeEDFC0\NKNdQF8SpfZJAKgjtXEXe23mp+CbBVMrbaqsLnsALGybnroVMATU5q16LgYBpuF+u+N\Rkq7Zxh6CZPukMQGQbOCUwNwO6BuumNhri5ayGjk1ZBiW82C7Nu0hs2pLcCgNwqMtt1+t+NW\85R6kyaco784272A4S0qQc5SwmXqQFBTRj+Wb0sVpGBLTIdy9Wg3pXeXN5n5o340dd22A\7EMQmqTC3c3v3f8B9ew17Qf3lmgeBvdMuoVDVe+p077IAWnGrgv1ZxY9sKEEqrCmau\4L5xSfbD02Kv8z2B18ZDcDn1RxZxZkNBXKJivsCVQDUxrazwGE0gqRmbk4F5yG5YCGC\41/O8Korun+jjP6wVQYKf1y1QUG31XKHXGUG1Fx0v3707bUjugycjVO5kGxho2zkgqog2z\ZEpkefvrnt6QncebW/a/4Qqji81JY8GD50mGVMhmduHgJtWGVJFJPBPinm+rem7t+0\4QGR7ywyRxcVt2i1WttPybfw+1XDN5NGseesigR1ycTrNnB80b16oxLwxtT4e0QO/L\UKasq7jJ3AQTHOofDdtpkqsthsFhpy+WjSquOXCyMYRc6pIMlx=xnJDF321ChLira\V6v\F2S0T5Y+WnNDESzTRnNhDUW5KAhau0K9WABVtcM5ba0u610a1NsfcrLag+KnhXnnH6P\oobcw/Jes1D57TA18Wwrqjth28x6h9eRRKerpaOw+FYk+2MyD901Y98VE12/EwOqAg11Pp\elAvueM01Rkp6g4bmsyYHtuNwGvc5qiy7/eGw9z2PV3J3g3jxvbyExqDFyDc076EK1\5WxDDFUzUjkfPkeCnxzQuIrq5s49BzcRyMt5ndEVE21TFqFZ/R8B7Rnxn2L1k+hqcc+cc\pEeAaqwgo1M1LcI/1CyY61md06Rhxwfl1H7Wge+jshN41kaeGh1019MGMTLbzq50cP8zP\6FLZKBTMWjWq90/Obj11MHR9EPblcE/E/VRIAZTUD62Df2tN6ep3Qvge/o6R1mE8TF\GwV1Jv9s56/08Dd61IHPKwBkF4f657hoKah2im0nyUpxowrRj1RMHCBT3n8c4\2Pf1+segSv3G1jps3tK157SPKQtTq16DMMwzEBdrtvdY1s3Hagpc1L1Diwa/yEF4RR2m3VNIJR9\6Nkx3tqen36Xmd/M1CkcdW5Rjcz/6ye1QWbr9CjIt/YLTQ2gtA1RISX90UDBVZ+E\E03V2Q+Ecsc97B5S9G/66CpZ1bk/HqAf+cu0YMbGnEM83nTu5ps454JABcw50XWNQ\D\FWEdguzULhhgq+hYExdbjv2LrElihkv8yq8a2+7RXXUZHJM/joNOGHHuOpUG/bfo3ktnaILC\xsOXMUBDC3VcitFFHsGK1svtcERDFxkh1AHA8gPa59j7Do6n3wEbnBDU1soKNFtpmcVKE

3.4.3. CRYDI Signature Representation

For the purpose of using the CRYSTALS-Dilithium Signature Algorithm (CRYDI) for signing data, it is recommended to use "JSON Web Signature (JWS)" [RFC7515], algorithm "CRYDI" is defined here, to be applied as the value of the "alg" parameter.

The following key subtypes are defined here for use with CRYDI:
The key type used with these keys is "PQK" and the algorithm used for signing is "CRYDI". These subtypes MUST NOT be used for key agreement.

The CRYDI variant used is determined by the subtype of the key (CRYDI3 for "pset 3" and CRYDI2 for "pset 2").

Implementations need to check that the key type is "PQK" for JOSE and that the pset of the key is a valid subtype when creating a signature.

The CRYDI digital signature is generated as follows:

1. Generate a digital signature of the JWS Signing Input using CRYDI with the desired private key, as described in Section 3.2 (#name-sign). The signature bit string is the concatenation of a bit packed representation of z and encodings of h and c in this order.

2. The resulting octet sequence is the JWS Signature.

When using a JWK for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "PQK" for JOSE.
* The "alg" field MUST be present, and it MUST represent the pset subtype.
* If the "key_ops" field is present, it MUST include "sign" when creating an CRYDI signature.
* If the "key_ops" field is present, it MUST include "verify" when verifying an CRYDI signature.
* If the JWK "use" field is present, its value MUST be "sig".
Example signature using only required fields, represented in compact form:

eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSIsIiI9.

SXTigJlzIGEgZGFuZ2Vyb3VzLCB0aGUgcm9mIHN3ZXB0IG9mZiB0by4

cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEnsk2I

The same example decoded for readability:

```
{ "header": { "alg": "CRYDI3", "kid": "did:example:123#key-0" },

"payload": "It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to." },

"signature": "2As8T1AHenWzLuTojcAYFDnT05n4bmDGIWenHxqVxIzL7311HtVg/\7PEJHYmpc1fFvNzm0xJt0asD5bQxk3ZY8WuEQDUJsn4j+zbyob8MPQI5u3p5zk1LhG/\6Q8p1qHd5v0y4a78VWnFXFJpYsEtc0DeECAft195zVh1jVjuDbPjW74ju/\DKambbJIDz/\NLYgYnNyPcqHj1fbP7acFeOgqBAQqvrWugvAkdeM+hU6djaXW25+FeU4LjluOIFBrjz/\ZJ04M07j7EmluhJWB374Q7Qr1fVqnvyr4z2a1LWHjjr7VenFZ2CIKpuRthSpNYWYTR3N3mM/\v00jVlyaqbJpmUuemhwhjaQc131P7c595KayGtyjLPEapsbn7yip1bod/R2ZPZ0eooool/\zD03VumsGrVLKj4OIFlnA8pgr+bJjisAPzWrGu6NBP8v4U2BD9Dr8wdzv/vl00Bwdg/\nfq1eX5l186e8tnd+21UCn4z2U+BKqo9R14VfEOt9g5aO1czWjJAN+0z8VqjZwzh/\z2toZNP4n2eWViF0PM3dpokUPE1391VH3QQ6V6TYfbMW9WQ6Uny1X2FEzNCNHF9Qhs/\7XehyEYDgJBFYIvbrTFCd+EBZbWQAnLkm7UXXXB7HdUsJMbTkwdffGw1ZWBTFUSG/\tcqFCPlbVxGbcCSe0XGh0CQKQwzwj3KNWY/\nhnhHbn7kyySqaI+w4ph3pKwrb38sCS/\Gb3ryptI8ze2OJr+1C1Wnu18onoJGjinCzq7jCGMlMCrFpZUV6rp/Fp1M261p28M9ShF/\BHsTN7KgsypIgG6Yc3Gz0J/4ir7V313yWguK71BuIUtM/OKwxtM75carZJPF/\21kn2Hh/\TC+JvBz/yaHs2r0d1rCwQosNh/a/c/b/c/mYmgHc3KQwRz/3AXr6we6urWJ8q6+v53QKK/\a1+CrLrKrVgJX6vh+ps1NB5EYvyMhBAXAb0JZ3k+dGed3G7v/\q1FkbnuI1U5f16LsH/\vAXsuVLtiTU6qrG0raTmvY35mXpuOUyCtNFB0+S73Ux5/N2dzrUpX1URW/\luFkCtanUimUomyoz6Yk02V34kh8qST8eqRBAoe8heYEktZSYov4YFcgkQHj9G6/\yc02uA33kwnM+J/g+hwIWAYlYF990KECTvFR8787TfGfn+3t2JWlnmHszPnSsTAI/\dNR+VmhNozpUZKQ0dGhuLftzAyAncamL38LHYHFHz2boaa0ysMGW8WtpE0Q3+BngoanNax/\dJ5THRmvhM53EDwAnErImsZ6j5k8uVhYtNiiKvbEWFyYoKk9U5MboEfdje4DX/\wIAefXYCgPK8eXhL+9qDLxAD1DQbcu+Ey3whX/r4r2Q6l+34HpRnn3g50k+Gto/3nig
```

3.5. Using CRYDI with COSE

The approach taken here matches the work done to support secp256k1 in JOSE and COSE in [RFC8812].

The following tables map terms between JOSE and COSE for signatures.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYDI5</td>
<td>TBD</td>
<td>TBD</td>
<td>No</td>
</tr>
<tr>
<td>CRYDI3</td>
<td>TBD</td>
<td>TBD</td>
<td>No</td>
</tr>
<tr>
<td>CRYDI2</td>
<td>TBD</td>
<td>TBD</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5

The following tables map terms between JOSE and COSE for key types.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQK</td>
<td>TBD</td>
<td>TBD</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6

4. Falcon

TODO

5. SPHINCS+

5.1. Overview

This section of the document describes the hash-based signature scheme SPHINCS+. The scheme is based on the concept of authenticating a large number or few-time signatures keypair using a combination of Markle-tree signatures, a so-called hypertree. For each message to be signed a (pseudo-)random FTS keypair is selected with which the message can be signed. Combining this signature along with an authentication path through the hyper-tree consisting of hash-based many-time signatures then gives the SPHINC+ signature. The parameter set is strategically chosen such that the probability of signing too many messages with a specific FTS keypair to impact security is small enough to prevent forgery attacks. A trade-off in parameter set can
be made on security guarantees, performance and signature size.

SPHINCS+ is a post-quantum approach to digital signatures that promises Post-Quantum Existential Unforgeability under Chosen Message Attack (PQ-EU-CMA), while ensuring that the security levels reached meet security needs for resistance to both classical and quantum attacks. The algorithm itself is based on the hardness assumptions of its underlying hash functions, which can be chosen from the set Haraka, SHA-256 or SHAKE256. For all security levels the only operations required are calls to these hash functions on various combinations of parameters and internal states.

Contrary to CRYSTALS-Dilithium and Falcon, SPHINCS+ is not based on any algebraic structure. This reduces the possible attack surface of the algorithm.

SPHINCS+ brings several advantages over other approaches to signature suites:

* Post Quantum in nature - use of cryptographically secure hash functions and other approaches that should remain hard problems even when under an attack utilizing quantum approaches
* Minimal security assumptions - compared to other schemes does not base its security on a new paradigm. The security is solely based on the security of the assumptions of the underlying hash function.
* Performance and Optimization - based on combining a great many hash function calls of SHA-256, SHAKE256 or Haraka means existing (secure) SW and HW implementations of those hash functions can be re-used for increased performance
* Private and Public Key Size - compared to other post quantum approaches a very small key size is the form of hash inputs-outputs. This then has the drawback that either a large signature or low signing speed has to be accepted
* Cryptanalysis assurance - attacks (both pre-quantum and quantum) are easy to relate to existing attacks on hash functions. This allows for precise quantification of the security levels
* Overlap with stateful hash-based algorithms - means there are possibilities to combine implementations with those of XMSS and LMS (TODO refs)
* Inherent resistance against side-channel attacks - since its core primitive is a hash function, it thereby is hard to attack with side-channels.
The primary known disadvantage to SPHINCS+ is the size signatures, or the speed of signing, depending on the chosen parameter set. Especially in IoT applications this might pose a problem. Additionally hash-based schemes are also vulnerable to differential and fault attacks.

5.2. Parameters

TODO

5.2.1. Parameter sets

TODO

5.3. Core Operations

TODO

5.3.1. Generate

TODO

5.3.2. Sign

TODO

5.3.3. Verify

TODO

5.4. Using SPHINCS+ with JOSE

Basing off of this (https://datatracker.ietf.org/doc/html/rfc8812#section-3)

5.4.1. SPHINCS+ Key Representations

TODO

5.4.2. SPHINCS+ Algorithms

TODO

5.4.2.1. Public Key

TODO
5.4.2.2. Private Key

TODO

5.4.3. SPHINCS+ Signature Representation

TODO

6. Security Considerations

The following considerations SHOULD apply to all signature schemes described in this specification, unless otherwise noted.

6.1. Validating public keys

All algorithms in that operate on public keys require first validating those keys. For the sign, verify and proof schemes, the use of KeyValidate is REQUIRED.

6.2. Side channel attacks

Implementations of the signing algorithm SHOULD protect the secret key from side-channel attacks. Multiple best practices exist to protect against side-channel attacks. Any implementation of the the CRYSTALS-Dilithium signing algorithm SHOULD utilize the following best practices at a minimum:

* Constant timing - the implementation should ensure that constant time is utilized in operations
* Sequence and memory access persistance - the implementation SHOULD execute the exact same sequence of instructions (at a machine level) with the exact same memory access independent of which polynomial is being operated on.
* Uniform sampling - uniform sampling is the default in CRYSTALS-Dilithium to prevent information leakage, however care should be given in implementations to preserve the property of uniform sampling in implementation.
* Secrecy of S1 - utmost care must be given to protection of S1 and to prevent information or power leakage. As is the case with most proposed lattice based approaches to date, forgery and other attacks may succeed, for example, with Dilithium through leakage of S1 (https://eprint.iacr.org/2018/821.pdf) through side channel mechanisms.

6.3. Randomness considerations

It is recommended that the all nonces are from a trusted source of randomness.
7. IANA Considerations

The following has NOT YET been added to the "JSON Web Key Types" registry:

* "kty" Parameter Value: "PQK"
* Key Type Description: Base 64 encoded string key pairs
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 2 of this document (TBD)

The following has NOT YET been added to the "JSON Web Key Parameters" registry:

* Parameter Name: "pset"
* Parameter Description: The parameter set of the crypto system
* Parameter Information Class: Public
* Used with "kty" Value(s): "PQK"
* Change Controller: IESG
* Specification Document(s): Section 2 of this document (TBD)

* Parameter Name: "xs"
* Parameter Description: The shake256 of the public key
* Parameter Information Class: Public
* Used with "kty" Value(s): "PQK"
* Change Controller: IESG
* Specification Document(s): Section 2 of this document (TBD)

* Parameter Name: "ds"
* Parameter Description: The shake256 of the private key
* Parameter Information Class: Private
* Used with "kty" Value(s): "PQK"
* Change Controller: IESG
* Specification Document(s): Section 2 of this document (TBD)

* Parameter Name: "d"
* Parameter Description: The private key
* Parameter Information Class: Private
* Used with "kty" Value(s): "PQK"
* Change Controller: IESG
* Specification Document(s): Section 2 of RFC 8037

* Parameter Name: "x"
* Parameter Description: The public key
* Parameter Information Class: Public
* Used with "kty" Value(s): "PQK"
* Change Controller: IESG
* Specification Document(s): Section 2 of RFC 8037

The following has NOT YET been added to the "JSON Web Signature and Encryption Algorithms" registry:
* Algorithm Name: "CRYDI3"
* Algorithm Description: CRYDI3 signature algorithms
* Algorithm Usage Location(s): "alg"
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 3.1 of this document (TBD)
* Algorithm Analysis Documents(s): (TBD)

The following has been added to the "JSON Web Key Lattice" registry:

* Lattice Name: "CRYDI5"
* Lattice Description: Dilithium 5 signature algorithm key pairs
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 3.1 of this document (TBD)
* Lattice Name: "CRYDI3"
* Lattice Description: Dilithium 3 signature algorithm key pairs
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 3.1 of this document (TBD)
* Lattice Name: "CRYDI2"
* Lattice Description: Dilithium 2 signature algorithm key pairs
* JOSE Implementation Requirements: Optional
* Change Controller: IESG
* Specification Document(s): Section 3.1 of this document (TBD)

8. Appendix

* JSON Web Key (JWK) - RFC7517 (https://tools.ietf.org/html/rfc7517)
* CRYSTALS-Dilithium - Dilithium (https://www.pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf)
8.1. Test Vectors

//TODO

9. Normative References

[CRYSTALS-Dilithium]
Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., and D. Stehle, "CRYSTALS-

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data
Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006,

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web
Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517,
DOI 10.17487/RFC7517, May 2015,

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK)
Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September

[RFC8702] Kampanakis, P. and Q. Dang, "Use of the SHAKE One-Way Hash
Functions in the Cryptographic Message Syntax (CMS)",
RFC 8702, DOI 10.17487/RFC8702, January 2020,

[RFC8812] Jones, M., "CBOR Object Signing and Encryption (COSE) and
JSON Object Signing and Encryption (JOSE) Registrations
for Web Authentication (WebAuthn) Algorithms", RFC 8812,
DOI 10.17487/RFC8812, August 2020,

10. Informative References

Authors’ Addresses

Michael Prorock
mesur.io
Email: mprorock@mesur.io

Orie Steele
Transmute
Email: orie@transmute.industries

Rafael Misoczki
Google
Email: rafaelmisoczki@google.com

Michael Osborne
IBM
Email: osb@zurich.ibm.com

Christine Cloostermans
NXP
Email: christine.cloostermans@nxp.com
Abstract

This document extends the CBOR Object Signing and Encryption (COSE) parameter kid to CBOR integer values.

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 20 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

Many Internet of Things (IoT) deployments require technologies which are highly performant in constrained environments [RFC7228]. The connectivity for these settings may exhibit extremely restricted bandwidth constraints, for which byte level optimizations are motivated, see [I-D.ietf-lake-reqs].

The use of CBOR [RFC8949] enables a compact encoding of protected data as COSE objects [I-D.ietf-cose/rfc8152bis-struct], which is a basic building block in various IoT security settings such as CWT [RFC8392], OSCORE [RFC8613], and ACE-OAuth [I-D.ietf-ace-oauth-authz]. COSE defines the key identifier parameter kid used to identify keys used in the COSE object.

The value of the kid parameter is specified to be encoded as a CBOR byte string, which (with the exception of the empty string) requires at least two bytes on the wire. For comparison, CBOR encoding of small integers (-24, ..., 23) need only one byte on the wire. Since many IoT deployments may use local identifiers for which a few unique identifiers are sufficient, the use of CBOR integers as key identifiers would reduce the overhead due to transport of COSE objects.

This specification amends this limitation by extending the COSE parameter kid to allow CBOR integer values. kid is used in different instances, which all need to be extended to CBOR int encoding:

* The kid COSE header parameter, see Section 3.1.
* The kid COSE Key Common Parameter, see Section 3.2.
* The kid CWT Confirmation Method, see Section 3.3.
2. Security Considerations

There are no additional security considerations compared to key identifiers to being byte strings.

3. IANA Considerations

3.1. COSE Header Parameters Registry

IANA has extended the Value Type of kid in the "COSE Header Parameters" registry under the group name "CBOR Object Signing and Encryption (COSE)" to also allow the Value Type int. The resulting Value Type is bstr / int. The Value Registry for this item is empty and omitted from the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kid</td>
<td>4</td>
<td>bstr / int</td>
<td>Key identifier</td>
</tr>
</tbody>
</table>

3.2. COSE Key Common Parameters Registry

IANA has extended the Value Type of kid in the "COSE Key Common Parameters" registry under the group name "CBOR Object Signing and Encryption (COSE)" to also allow the Value Type int. The resulting Value Type is bstr / int. The Value Registry for this item is empty and omitted from the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kid</td>
<td>2</td>
<td>bstr / int</td>
<td>Key identification value - match to kid in message</td>
</tr>
</tbody>
</table>

3.3. CWT Confirmation Methods

IANA has extended the Value Type of kid in the "CWT Confirmation Methods" registry under the group name "CBOR Web Token (CWT) Claims" to also allow the Value Type int. The resulting Value Type is bstr / int. The Value Registry for this item is empty and omitted from the table below.
<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kid</td>
<td>3</td>
<td>bstr / int</td>
<td>Key identification value - match to kid in message</td>
</tr>
</tbody>
</table>

4. References

4.1. Normative References

[I-D.ietf-cose-rfc8152bis-struct]


4.2. Informative References

[I-D.ietf-ace-oauth-authz]

[I-D.ietf-lake-reqs]


Acknowledgments

Authors’ Addresses

Göran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden
Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden
Email: john.mattsson@ericsson.com