SUIT
Internet-Draft
Intended status: Standards Track
Expires: 23 January 2026

B. Moran
Arm Limited
. Rnningstad
Nordic Semiconductor
A. Tsukamoto
Openchip & Software Technologies, S.L.

22 July 2025

Cryptographic Algorithms for Internet of Things (IoT) Devices
draft-ietf-suit-mti-23

Abstract

The SUIT manifest, as defined in "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices" (RFC 9124), provides a flexible and extensible format for describing how firmware and software updates are to be fetched, verified, decrypted, and installed on resource-constrained devices. To ensure the security of these update processes, the manifest relies on cryptographic algorithms for functions such as digital signature verification, integrity checking, and confidentiality.

This document defines cryptographic algorithm profiles for use with the Software Updates for Internet of Things (SUIT) manifest. These profiles specify sets of algorithms to promote interoperability across implementations.

Given the diversity of IoT deployments and the evolving cryptographic landscape, algorithm agility is essential. This document groups algorithms into named profiles to accommodate varying levels of device capabilities and security requirements. These profiles support the use cases laid out in the SUIT architecture, published in "A Firmware Update Architecture for Internet of Things" (RFC 9019).

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/.

Copyright Notice

Table of Contents

1. Introduction

This document defines algorithm profiles, in IANA registry (Section 6), intended for authors of Software Updates for Internet of Things (SUIT) manifests and their recipients, with the goal of promoting interoperability in software update scenarios for constrained nodes. These profiles specify sets of algorithms that are tailored to the evolving security landscape, recognizing that cryptographic requirements may change over time.

The following profiles are defined:

* One profile designed for constrained devices that support only symmetric key cryptography

* Two profiles for constrained devices capable of using asymmetric key cryptography

* Two profiles that employ Authenticated Encryption with Associated Data (AEAD) ciphers

* One constrained asymmetric profile that uses a hash-based signature scheme

Due to the asymmetric nature of SUIT deployments - where manifest authors typically operate in resource-rich environments while recipients are resource-constrained - the cryptographic requirements differ between these two roles.

This specification uses AES-CTR in combination with a digest algorithm, as defined in [RFC9459], to support use cases that require out-of-order block reception and decryption-capabilities not offered by AEAD algorithms. For further discussion of these constrained use cases, refer to Section 4.2. Other SUIT use cases (see [I-D.ietf-suit-manifest]) may define different profiles.

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.

This specification uses the following abbreviations:

* Advanced Encryption Standard (AES)

* AES Counter (AES-CTR) Mode

* AES Key Wrap (AES-KW)

* Authenticated Encryption with Associated Data (AEAD)

* Concise Binary Object Representation (CBOR)

* CBOR Object Signing and Encryption (COSE)

* Concise Data Definition Language (CDDL)

* Elliptic Curve Diffie-Hellman Ephemeral-Static (ECDH-ES)

* Hash-based Message Authentication Code (HMAC)

* Hierarchical Signature System / Leighton-Micali Signature (HSS/LMS)

* Software Updates for Internet of Things (SUIT)

SUIT specifically addresses the requirements of constrained devices and networks, as described in [RFC9019].

The terms "Author", "Recipient", and "Manifest" are defined in [I-D.ietf-suit-manifest].

3. Profiles

Each profile, in IANA registry (Section 6), consists of algorithms from the following categories:

* Digest Algorithms

* Authentication Algorithms

* Key Exchange Algorithms (optional)

* Encryption Algorithms (optional)

Each profile references specific algorithm identifiers, as defined in [IANA-COSE]. Since these algorithm identifiers are used in the context of the IETF SUIT manifest [I-D.ietf-suit-manifest], they are represented using CBOR Object Signing and Encryption (COSE) structures as defined in [RFC9052] and [RFC9053].

The use of the profiles by authors and recipients is based on the following assumptions:

* Recipients MAY choose which profile they wish to implement. It is RECOMMENDED that they implement the suit-sha256-hsslms-a256kw-a256ctr profile (Section 3.6). Recipients MAY implement any number of other profiles not defined in this document. Recipients MAY choose not to implement encryption and the corresponding key exchange algorithms if they do not intend to support encrypted payloads.

* Authors MUST implement all profiles with a status set to 'MANDATORY' in Section 6. Authors MAY implement any number of additional profiles.

3.1. Profile suit-sha256-hmac-a128kw-a128ctr

This profile only offers support for symmetric cryptographic algorithms.

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	HMAC-256	5
Key Exchange	A128KW Key Wrap	-3
Encryption	A128CTR	-65534

Table 1

3.2. Profile suit-sha256-esp256-ecdh-a128ctr

This profile supports asymmetric algorithms for use with constrained devices.

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	ESP256	-9
Key Exchange	ECDH-ES + A128KW	-29
Encryption	A128CTR	-65534

Table 2

3.3. Profile suit-sha256-ed25519-ecdh-a128ctr

This profile supports an alternative choice of asymmetric algorithms for use with constrained devices.

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	Ed25519	-19
Key Exchange	ECDH-ES + A128KW	-29
Encryption	A128CTR	-65534

Table 3

3.4. Profile suit-sha256-esp256-ecdh-a128gcm

This profile supports asymmetric algorithms in combination with AEAD algorithms.

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	ESP256	-9
Key Exchange	ECDH-ES + A128KW	-29
Encryption	A128GCM	1

Table 4

3.5. Profile suit-sha256-ed25519-ecdh-chacha-poly

This profile also supports asymmetric algorithms with AEAD algorithms but offers an alternative to suit-sha256-esp256-ecdh-a128gcm.

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	Ed25519	-19
Key Exchange	ECDH-ES + A128KW	-29
Encryption	ChaCha20/Poly1305	24

Table 5

3.6. Profile suit-sha256-hsslms-a256kw-a256ctr

This profile utilizes a stateful hash-based signature algorithm, namely the Hierarchical Signature System / Leighton-Micali Signature (HSS/LMS), as a unique alternative to the profiles listed above.

A note regarding the use of the HSS/LMS: The decision as to how deep the tree is, is a decision that affects authoring tools only (see [RFC8778]). Verification is not affected by the choice of the "W" parameter, but the size of the signature is affected. To support the long lifetimes required by IoT devices, it is RECOMMENDED to use trees with greater height (see Section 2.2 of [RFC8778]).

Algorithm Type	Algorithm	COSE Key
Digest	SHA-256	-16
Authentication	HSS/LMS	-46
Key Exchange	A256KW	-5
Encryption	A256CTR	-65532

Table 6

4. Security Considerations

Payload encryption is used to protect sensitive content such as machine learning models, proprietary algorithms, and personal data [RFC6973]. In the context of SUIT, the primary purpose of payload encryption is to defend against unauthorized observation during distribution. By encrypting the payload, confidential information can be safeguarded from eavesdropping.

However, encrypting firmware or software update payloads on commodity devices do not constitute an effective cybersecurity defense against targeted attacks. Once an attacker gains access to a device, they may still be able to extract the plaintext payload.

4.1. Payload Encryption as Part of a Defense-in-Depth Strategy

To define the purpose of payload encryption as a defensive cybersecurity tool, it is important to define the capabilities of modern threat actors. A variety of capabilities are possible:

* find bugs by binary code inspection
* send unexpected data to communication interfaces, looking for unexpected behavior
* use fault injection to bypass or manipulate code
* use communication attacks or fault injection along with gadgets found in the code

Given this range of capabilities, it is important to understand which capabilities are impacted by firmware encryption. Threat actors who find bugs by manual inspection or use gadgets found in the code will need to first extract the code from the target. In the IoT context, it is expected that most threat actors will start with sample devices and physical access to test attacks.

Due to these factors, payload encryption serves to limit the pool of attackers to those who have the technical capability to extract code from physical devices and those who perform code-free attacks.

4.2. Use of AES-CTR in Payload Encryption

AES-CTR mode with a digest is specified, see [RFC9459]. All of the AES-CTR security considerations in [RFC9459] apply. See [I-D.ietf-suit-firmware-encryption] for additional background information.

5. Operational Considerations

While this document focuses on the cryptographic aspects of manifest processing, several operational and manageability considerations are relevant when deploying these profiles in practice.

5.1. Profile Support Discovery

To enable interoperability of the described profiles, it is important for a manifest author to determine which profiles are supported by a device. Furthermore, it is also important for the author and the distribution system (see Section 3 of [I-D.ietf-suit-firmware-encryption]) to know whether firmware for a particular device or family of devices needs to be encrypted, and which key distribution mechanism shall be used. This information can be obtained through:

* Manual configuration.

* Device management systems, as described in [RFC9019], which typically maintain metadata about device capabilities and their lifecycle status. These systems may use proprietary or standardized management protocols to expose supported features. LwM2M [LwM2M] is one such standardized protocol. The Trusted Execution Environment Provisioning (TEEP) protocol [I-D.ietf-teep-protocol] is another option.

* Capability reporting mechanisms, such as those described in [I-D.ietf-suit-report], which define structures that allow a device to communicate supported SUIT features and cryptographic capabilities to a management or attestation entity.

5.2. Profile Selection and Control

When a device supports multiple algorithm profiles, it is expected that the SUIT manifest author indicates the appropriate profile based on the intended recipient(s) and other policies. The manifest itself indicates which algorithms are used; devices are expected to validate manifests using supported algorithms.

Devices do not autonomously choose which profile to apply; rather, they either accept or reject a manifest based on the algorithm profile it uses. There is no protocol-level negotiation of profiles at SUIT manifest processing time. Any dynamic profile selection or configuration is expected to occur as part of other protocols, for example, through device management.

5.3. Profile Provisioning and Constraints

Provisioning for a given profile may include:

* Installation of trust anchors for acceptable signers.

* Distribution of keys used by the content key distribution mechanism (see Section 4 of [I-D.ietf-suit-firmware-encryption]).

* Availability of specific cryptographic libraries or hardware support (e.g., for post-quantum algorithms or AEAD ciphers).

* Evaluation of the required storage and processing resources for the selected profile.

* Support for manifest processing capabilities.

There may be conditions under which switching to a different algorithm profile is not feasible, such as:

* Lack of hardware support (e.g., no crypto acceleration).

* Resource limitations on memory-constrained devices (e.g., insufficient flash or RAM).

* Deployment policy constraints or regulatory compliance requirements.

In such cases, a device management or update orchestration system should take these constraints into account when constructing and distributing manifests.

5.4. Logging and Reporting

Implementations MAY log failures to process a manifest due to unsupported algorithm profiles or unavailable cryptographic functionality. When supported, such events SHOULD be reported using secure mechanisms, such as those described in [I-D.ietf-suit-report], to assist operators in diagnosing update failures or misconfigurations.

6. IANA Considerations

IANA is requested to create a new "COSE SUIT Algorithm Profiles" registry, to be located within its own self-titled registry group. The registry will be listed in the "Software Update for the Internet of Things (SUIT)" category at https://www.iana.org/protocols.

While most profile attributes are self-explanatory, the status field warrants a brief explanation. This field can take one of three values: MANDATORY, NOT RECOMMENDED, or OPTIONAL.

* MANDATORY indicates that the profile is mandatory to implement for manifest authors.

* NOT RECOMMENDED means that the profile should generally be avoided in new implementations.

* OPTIONAL suggests that support for the profile is permitted but not required.

IANA is requested to add a note that mirrors these status values to the registry.

Adding new profiles or updating the status of existing profiles requires Standards Action (Section 4.9 of [RFC8126]).

As time progresses, algorithm profiles may lose their MANDATORY status. When this occurs, their status may be changed to either OPTIONAL or NOT RECOMMENDED for new implementations. Similarly, a profile may be transitioned from OPTIONAL to NOT RECOMMENDED. However, profiles once marked as OPTIONAL or NOT RECOMMENDED MUST NOT be transitioned to MANDATORY status in future revisions. Since it may be impossible to update certain parts of IoT device firmware in the field, such as first-stage bootloaders, support for all relevant algorithms will almost always be required by authoring tools.

The initial content of the "COSE SUIT Algorithm Profiles" registry is:

6.1. Profile: suit-sha256-hmac-a128kw-a128ctr

* Profile: suit-sha256-hmac-a128kw-a128ctr
* Status: MANDATORY
* Digest: -16
* Auth: 5
* Key Exchange: -3
* Encryption: -65534
* Descriptor Array: [-16, 5, -3, -65534]
* Reference: Section 3.1 of THIS_DOCUMENT

6.2. Profile: suit-sha256-esp256-ecdh-a128ctr

* Profile: suit-sha256-esp256-ecdh-a128ctr
* Status: MANDATORY
* Digest: -16
* Auth: -9
* Key Exchange: -29
* Encryption: -65534
* Descriptor Array: [-16, -9, -29, -65534]
* Reference: Section 3.2 of THIS_DOCUMENT

6.3. Profile: suit-sha256-ed25519-ecdh-a128ctr

* Profile: suit-sha256-ed25519-ecdh-a128ctr
* Status: MANDATORY
* Digest: -16
* Auth: -19

* Key Exchange: -29
* Encryption: -65534
* Descriptor Array: [-16, -19, -29, -65534]
* Reference: Section 3.3 of THIS_DOCUMENT

6.4. Profile: suit-sha256-esp256-ecdh-a128gcm

* Profile: suit-sha256-esp256-ecdh-a128gcm
* Status: MANDATORY
* Digest: -16
* Auth: -9
* Key Exchange: -29
* Encryption: 1
* Descriptor Array: [-16, -9, -29, 1]
* Reference: Section 3.4 of THIS_DOCUMENT

6.5. Profile: suit-sha256-ed25519-ecdh-chacha-poly

* Profile: suit-sha256-ed25519-ecdh-chacha-poly
* Status: MANDATORY
* Digest: -16
* Auth: -19
* Key Exchange: -29
* Encryption: 24
* Descriptor Array: [-16, -19, -29, 24]
* Reference: Section 3.5 of THIS_DOCUMENT

6.6. Profile: suit-sha256-hsslms-a256kw-a256ctr

* Profile: suit-sha256-hsslms-a256kw-a256ctr

* Status: MANDATORY

* Digest: -16

* Auth: -46

* Key Exchange: -5

* Encryption: -65532

* Descriptor Array: [-16, -46, -5, -65532]

* Reference: Section 3.6 of THIS_DOCUMENT

7. References

7.1. Normative References

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rnningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-34, 28 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-34>.

[IANA-COSE]
"CBOR Object Signing and Encryption (COSE)", 2022, <https://www.iana.org/assignments/cose/cose.xhtml>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8778] Housley, R., "Use of the HSS/LMS Hash-Based Signature Algorithm with CBOR Object Signing and Encryption (COSE)", RFC 8778, DOI 10.17487/RFC8778, April 2020, <https://www.rfc-editor.org/rfc/rfc8778>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[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>.

[RFC9459] Housley, R. and H. Tschofenig, "CBOR Object Signing and Encryption (COSE): AES-CTR and AES-CBC", RFC 9459, DOI 10.17487/RFC9459, September 2023, <https://www.rfc-editor.org/rfc/rfc9459>.

7.2. Informative References

[I-D.ietf-suit-firmware-encryption] Tschofenig, H., Housley, R., Moran, B., Brown, D., and K. Takayama, "Encrypted Payloads in SUIT Manifests", Work in Progress, Internet-Draft, draft-ietf-suit-firmware-encryption-25, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-firmware-encryption-25>.

[I-D.ietf-suit-report] Moran, B. and H. Birkholz, "Secure Reporting of Update Status", Work in Progress, Internet-Draft, draft-ietf-suit-report-14, 22 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-report-14>.

[I-D.ietf-teep-protocol]
Tschofenig, H., Pei, M., Wheeler, D. M., Thaler, D., and A. Tsukamoto, "Trusted Execution Environment Provisioning (TEEP) Protocol", Work in Progress, Internet-Draft, draft-ietf-teep-protocol-21, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-teep-protocol-21>.

[LwM2M] Open Mobile Alliance, "OMA Lightweight M2M", 20 April 2025, <https://www.openmobilealliance.org/specifications/lwm2m>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

Appendix A. Full CDDL

The following CDDL snippet [RFC8610] creates a subset of COSE for use with SUIT. Both tagged and untagged messages are defined. SUIT only uses tagged COSE messages, but untagged messages are also defined for use in protocols that share a ciphersuite with SUIT.

To be valid, the following CDDL MUST have the COSE CDDL appended to it. The COSE CDDL can be obtained by following the directions in [RFC9053], Section 1.4.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

SUIT_COSE_tool_tweak /= suit-sha256-hmac-a128kw-a128ctr
SUIT_COSE_tool_tweak /= suit-sha256-esp256-ecdh-a128ctr
SUIT_COSE_tool_tweak /= suit-sha256-ed25519-ecdh-a128ctr
SUIT_COSE_tool_tweak /= suit-sha256-esp256-ecdh-a128gcm
SUIT_COSE_tool_tweak /= suit-sha256-ed25519-ecdh-chacha-poly
SUIT_COSE_tool_tweak /= suit-sha256-hsslms-a256kw-a256ctr
SUIT_COSE_tool_tweak /= SUIT_COSE_Profiles

SUIT_COSE_Profiles /= SUIT_COSE_Profile_HMAC_A128KW_A128CTR
SUIT_COSE_Profiles /= SUIT_COSE_Profile_ESP256_ECDH_A128CTR
SUIT_COSE_Profiles /= SUIT_COSE_Profile_ED25519_ECDH_A128CTR
SUIT_COSE_Profiles /= SUIT_COSE_Profile_ESP256_ECDH_A128GCM
```

```
   SUIT_COSE_Profiles /= \
                         SUIT_COSE_Profile_ED25519_ECDH_CHACHA20_POLY1304
   SUIT_COSE_Profiles /= SUIT_COSE_Profile_HSSLMS_A256KW_A256CTR

   suit-sha256-hmac-a128kw-a128ctr    = [-16, 5, -3, -65534]
   suit-sha256-esp256-ecdh-a128ctr     = [-16, -9, -29, -65534]
   suit-sha256-ed25519-ecdh-a128ctr     = [-16, -19, -29, -65534]
   suit-sha256-esp256-ecdh-a128gcm     = [-16, -9, -29, 1]
   suit-sha256-ed25519-ecdh-chacha-poly = [-16, -19, -29, 24]
   suit-sha256-hsslms-a256kw-a256ctr  = [-16, -46, -5, -65532]

   SUIT_COSE_Profile_HMAC_A128KW_A128CTR =
       SUIT_COSE_Profile<5,-65534> .and COSE_Messages
   SUIT_COSE_Profile_ESP256_ECDH_A128CTR =
       SUIT_COSE_Profile<-9,-65534> .and COSE_Messages
   SUIT_COSE_Profile_ED25519_ECDH_A128CTR =
       SUIT_COSE_Profile<-19,-65534> .and COSE_Messages
   SUIT_COSE_Profile_ESP256_ECDH_A128GCM =
       SUIT_COSE_Profile<-9,1> .and COSE_Messages
   SUIT_COSE_Profile_ED25519_ECDH_CHACHA20_POLY1304 =
       SUIT_COSE_Profile<-19,24> .and COSE_Messages
   SUIT_COSE_Profile_HSSLMS_A256KW_A256CTR =
       SUIT_COSE_Profile<-46,-65532> .and COSE_Messages

   SUIT_COSE_Profile<authid, encid> = SUIT_COSE_Messages<authid,encid>

   SUIT_COSE_Messages<authid, encid> =
       SUIT_COSE_Untagged_Message<authid, encid> /
       SUIT_COSE_Tagged_Message<authid, encid>

   SUIT_COSE_Untagged_Message<authid, encid> = SUIT_COSE_Sign<authid> /
       SUIT_COSE_Sign1<authid> / SUIT_COSE_Encrypt<encid> /
       SUIT_COSE_Encrypt0<encid> / SUIT_COSE_Mac<authid> /
       SUIT_COSE_Mac0<authid>

   SUIT_COSE_Tagged_Message<authid, encid> =
       SUIT_COSE_Sign_Tagged<authid> / SUIT_COSE_Sign1_Tagged<authid> /
       SUIT_COSE_Encrypt_Tagged<encid> / SUIT_COSE_Encrypt0_Tagged<\
                                                              encid> /
       SUIT_COSE_Mac_Tagged<authid> / SUIT_COSE_Mac0_Tagged<authid>

   ; Note: This is not the same definition as is used in COSE.
   ; It restricts a COSE header definition further without
   ; repeating the COSE definition. It should be merged
   ; with COSE by using the CDDL .and operator.
   SUIT_COSE_Profile_Headers<algid> = (
       protected : bstr .cbor SUIT_COSE_alg_map<algid>,
       unprotected : SUIT_COSE_header_map
```

```
)
SUIT_COSE_alg_map<algid> = {
    1 => algid,
    * int => any
}

SUIT_COSE_header_map = {
    * int => any
}

SUIT_COSE_Sign_Tagged<authid> = #6.98(SUIT_COSE_Sign<authid>)


SUIT_COSE_Sign<authid> = [
    SUIT_COSE_Profile_Headers<authid>,
    payload : bstr / nil,
    signatures : [+ SUIT_COSE_Signature<authid>]
]


SUIT_COSE_Signature<authid> =  [
    SUIT_COSE_Profile_Headers<authid>,
    signature : bstr
]


SUIT_COSE_Sign1_Tagged<authid> = #6.18(SUIT_COSE_Sign1<authid>)


SUIT_COSE_Sign1<authid> = [
    SUIT_COSE_Profile_Headers<authid>,
    payload : bstr / nil,
    signature : bstr
]


SUIT_COSE_Encrypt_Tagged<encid> = #6.96(SUIT_COSE_Encrypt<encid>)


SUIT_COSE_Encrypt<encid> = [
    SUIT_COSE_Profile_Headers<encid>,
    ciphertext : bstr / nil,
    recipients : [+SUIT_COSE_recipient<encid>]
]


SUIT_COSE_recipient<encid> = [
    SUIT_COSE_Profile_Headers<encid>,
```

```
        ciphertext : bstr / nil,
        ? recipients : [+SUIT_COSE_recipient<encid>]
    ]


    SUIT_COSE_Encrypt0_Tagged<encid> = #6.16(SUIT_COSE_Encrypt0<encid>)


    SUIT_COSE_Encrypt0<encid> = [
        SUIT_COSE_Profile_Headers<encid>,
        ciphertext : bstr / nil,
    ]


    SUIT_COSE_Mac_Tagged<authid> = #6.97(SUIT_COSE_Mac<authid>)


    SUIT_COSE_Mac<authid> = [
       SUIT_COSE_Profile_Headers<authid>,
       payload : bstr / nil,
       tag : bstr,
       recipients :[+SUIT_COSE_recipient<authid>]
    ]


    SUIT_COSE_Mac0_Tagged<authid> = #6.17(SUIT_COSE_Mac0<authid>)


    SUIT_COSE_Mac0<authid> = [
       SUIT_COSE_Profile_Headers<authid>,
       payload : bstr / nil,
       tag : bstr,
    ]
```

Appendix B. Acknowledgments

We would like to specifically thank Henk Birkholz, Mohamed Boucadair, Deb Cooley, Lorenzo Corneo, Linda Dunbar, Russ Housley, Michael B. Jones, Jouni Korhonen, Magnus Nystrm, Michael Richardson, and Hannes Tschofenig for their review comments.

Authors' Addresses

Brendan Moran
Arm Limited
Email: brendan.moran.ietf@gmail.com

yvind Rnningstad
Nordic Semiconductor
Email: oyvind.ronningstad@gmail.com

Akira Tsukamoto
Openchip & Software Technologies, S.L.
Email: akira.tsukamoto@gmail.com