Stream: Internet Engineering Task Force (IETF)
RFC: 9338
STD: 96
Updates: 9052
Category: Standards Track
Published: December 2022
ISSN: 2070-1721
Author: J. Schaad
August Cellars

RFC 9338
CBOR Object Signing and Encryption (COSE): Countersignatures

Abstract

Concise Binary Object Representation (CBOR) is a data format designed for small code size and small message size. CBOR Object Signing and Encryption (COSE) defines a set of security services for CBOR. This document defines a countersignature algorithm along with the needed header parameters and CBOR tags for COSE. This document updates RFC 9052.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9338.

Table of Contents

1. Introduction

There has been an increased focus on small, constrained devices that make up the Internet of Things (IoT). One of the standards that has come out of this process is "Concise Binary Object Representation (CBOR)" [RFC8949]. CBOR extended the data model of the JavaScript Object Notation (JSON) [STD90] by allowing for binary data, among other changes. CBOR has been adopted by several of the IETF working groups dealing with the IoT world as their method of encoding data structures. CBOR was designed specifically to be small in terms of both messages transported and implementation size and to have a schema-free decoder. A need exists to provide message security services for IoT, and using CBOR as the message-encoding format makes sense.

A countersignature is a second signature that confirms the primary signature. During the process of advancing CBOR Object Signing and Encryption (COSE) to Internet Standard, it was noticed that the description of the security properties of countersignatures was incorrect for the COSE_Sign1 structure mentioned in Section 4.5 of [RFC8152]. To remedy this situation, the COSE Working Group decided to remove all of the countersignature text from [RFC9052], which obsoletes [RFC8152]. This document defines a new countersignature with the desired security properties.

The problem with the previous countersignature algorithm was that the cryptographically computed value was not always included. The initial assumption that the cryptographic value was in the third slot of the array was known not to be true at the time, but in the case of the Message Authentication Code (MAC) structures this was not deemed to be an issue. The new algorithm defined in this document requires the inclusion of more values for the countersignature computation. The exception to this is the COSE_Signature structure where there is no cryptographically computed value.

The new algorithm defined in this document is designed to produce the same countersignature value in those cases where the computed cryptographic value was already included. This means that for those structures the only thing that would need to be done is to change the value of the header parameter.

With the publication of this document, implementers are encouraged to migrate uses of the previous countersignature algorithm to the one specified in this document. In particular, uses of "CounterSignature" will migrate to "CounterSignatureV2", and uses of "CounterSignature0" will migrate to "CounterSignature0V2". In addition, verification of "CounterSignature" must be supported by new implementations to remain compatible with senders that adhere to [RFC8152], which assumes that all implementations will understand "CounterSignature" as header parameter label 7. Likewise, verification of "CounterSignature0" may be supported for further compatibility.

1.1. Requirements 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.

1.2. CBOR Grammar

CBOR grammar in this document uses the Concise Data Definition Language (CDDL) [RFC8610].

The collected CDDL can be extracted from the XML version of this document via the XPath expression below. (Depending on the XPath evaluator one is using, it may be necessary to deal with > as an entity.)

```
//sourcecode[@type='cddl']/text()
```

CDDL expects the initial non-terminal symbol to be the first symbol in the file. For this reason, the first fragment of CDDL is presented here.

```
start = COSE_Countersignature_Tagged / Internal_Types

; This is defined to make the tool quieter:
Internal_Types = Countersign_structure / COSE_Countersignature0
```

The non-terminal Internal_Types is defined for dealing with the automated validation tools used during the writing of this document. It references those non-terminals that are used for security computations but are not emitted for transport.

1.3. Document Terminology

In this document, we use the following terminology.

"Byte" is a synonym for "octet".

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use in constrained systems. It is defined in [RFC7252].

"Context" is used throughout this document to represent information that is not part of the COSE message. Information that is part of the context can come from different sources, including protocol interactions, associated key structures, and application configuration. The context to use can be implicit, identified using either the "kid context" header parameter defined in [RFC8613] or a protocol-specific identifier. Context should generally be included in the cryptographic construction; for more details, see Section 4.4 of [RFC9052].

The term "byte string" is used for sequences of bytes, while the term "text string" is used for sequences of characters.

2. Countersignature Header Parameters

This section defines a set of common header parameters. A summary of these header parameters can be found in Table 1. This table should be consulted to determine the value of the label and the type of the value.

The set of header parameters defined in this section is:

V2 countersignature: This header parameter holds one or more countersignature values. Countersignatures provide a method of having a second party sign some data. The countersignature header parameter can occur as an unprotected attribute in any of the following structures that are defined in [RFC9052]: COSE_Sign1, COSE_Signature, COSE_Encrypt, COSE_recipient, COSE_Encrypt0, COSE_Mac, and COSE_Mac0. Details of version 2 countersignatures are found in Section 3.

Name	Label	Value Type	Description
Countersignature version 2	11	COSE_Countersignature / [+ COSE_Countersignature]	V2 countersignature attribute
Countersignature0 version 2	12	COSE_Countersignature0	V2 Abbreviated Countersignature

Table 1: Common Header Parameters

The CDDL fragment that represents the set of header parameters defined in this section is given below. Each of the header parameters is tagged as optional because they do not need to be in every map; however, the header parameters required in specific maps are discussed above.

```
CountersignatureV2_header = (
    ? 11 => COSE_Countersignature / [+ COSE_Countersignature]
)

Countersignature0V2_header = (
    ? 12 => COSE_Countersignature0
)
```

3. Version 2 Countersignatures

A countersignature is normally defined as a second signature that confirms a primary signature. A normal example of a countersignature is the signature that a notary public places on a document as witnessing that you have signed the document. A notary typically includes a timestamp to indicate when notarization occurs; however, such a timestamp has not yet been defined for COSE. A timestamp, once defined in a future document, might be included as a protected header parameter. Thus, applying a countersignature to either the COSE_Signature or COSE_Sign1 objects matches this traditional definition. This document extends the context of a countersignature to allow it to be applied to all of the security structures defined. The countersignature needs to be treated as a separate operation from the initial operation even if it is applied by the same user, as is done in [GROUP-OSCORE].

COSE supports two different forms for countersignatures. Full countersignatures use the structure COSE_Countersignature, which has the same structure as COSE_Signature. Thus, full countersignatures can have protected and unprotected attributes, including chained countersignatures. Abbreviated countersignatures use the structure COSE_Countersignature0. This structure only contains the signature value and nothing else. The structures cannot be converted between each other; as the signature computation includes a parameter identifying which structure is being used, the converted structure will fail signature validation.

The version 2 countersignature changes the algorithm used for computing the signature from the original version that is specified in Section 4.5 of [RFC8152]. The new version now includes the cryptographic material generated for all of the structures rather than just for a subset.

COSE was designed for uniformity in how the data structures are specified. One result of this is that for COSE one can expand the concept of countersignatures beyond just the idea of signing a signature to being able to sign most of the structures without having to create a new signing layer. When creating a countersignature, one needs to be clear about the security properties that result. When done on a COSE_Signature or COSE_Sign1, the normal countersignature semantics are preserved. That is, the countersignature makes a statement about the existence of a signature and, when used with a yet-to-be-specified timestamp, a point in time at which the signature exists. When done on a COSE_Mac or COSE_Mac0, the payload is included as well as the MAC value. When done on a COSE_Encrypt or COSE_Encrypt0, the existence of the encrypted data is attested to. It should be noted that there is a distinction between attesting to the encrypted data as opposed to attesting to the unencrypted data. If the latter is what is desired, then one needs to apply a signature to the data and then encrypt that. It is always possible to construct cases where

the use of two different keys results in successful decryption, where the tag check succeeds, but two completely different plaintexts are produced. This situation is not detectable by a countersignature on the encrypted data.

3.1. Full Countersignatures

The COSE_Countersignature structure allows for the same set of capabilities as a COSE_Signature. This means that all of the capabilities of a signature are duplicated with this structure. Specifically, the countersigner does not need to be related to the producer of what is being countersigned, as key and algorithm identification can be placed in the countersignature attributes. This also means that the countersignature can itself be countersigned. This is a feature required by protocols such as long-term archiving services. More information on how countersignatures are used can be found in the evidence record syntax described in [RFC4998].

The full countersignature structure can be encoded as either tagged or untagged, depending on the context. A tagged COSE_Countersignature structure is identified by the CBOR tag 19. The countersignature structure is the same as that used for a signer on a signed object. The CDDL fragment for full countersignatures is:

```
COSE_Countersignature_Tagged = #6.19(COSE_Countersignature)
COSE_Countersignature = COSE_Signature
```

The details of the fields of a countersignature can be found in Section 4.1 of [RFC9052].

An example of a countersignature on a signature can be found in Appendix A.1.1. An example of a countersignature in an encryption object can be found in Appendix A.3.1.

It should be noted that only a signature algorithm with appendix (see Section 8.1 of [RFC9052]) can be used for countersignatures. This is because the body should be able to be processed without having to evaluate the countersignature, and this is not possible for signature schemes with message recovery.

3.2. Abbreviated Countersignatures

Abbreviated countersignatures support encrypted group messaging where identification of the message originator is required but there is a desire to keep the countersignature as small as possible. For abbreviated countersignatures, there is no provision for any protected attributes related to the signing operation. That is, the parameters for computing or verifying the abbreviated countersignature are provided by the same context used to describe the encryption, signature, or MAC processing.

The CDDL fragment for the abbreviated countersignatures is:

```
COSE_Countersignature0 = bstr
```

The byte string representing the signature value is placed in the Countersignature0 attribute. This attribute is then encoded as an unprotected header parameter.

3.3. Signing and Verification Process

In order to create a signature, a well-defined byte string is needed. The Countersign_structure is used to create the canonical form. This signing and verification process takes in the countersignature target structure (COSE_Signature, COSE_Sign1, COSE_Sign, COSE_Mac, COSE_Mac0, COSE_Encrypt, or COSE_Encrypt0), the signer information (COSE_Signature), and the application data (external source). A Countersign_structure is a CBOR array. The target structure of the countersignature needs to have all of its cryptographic functions finalized before computing the signature. The fields of the Countersign_structure, in order, are:

context: A context text string identifying the context of the signature. The context text string is one of the following:

- "CounterSignature" for countersignatures using the COSE_Countersignature structure when other_fields is absent.
- "CounterSignature0" for countersignatures using the COSE_Countersignature0 structure when other_fields is absent.
- "CounterSignatureV2" for countersignatures using the COSE_Countersignature structure when other_fields is present.
- "CounterSignature0V2" for countersignatures using the COSE_Countersignature0 structure when other_fields is present.

body_protected: The serialized protected attributes from the target structure, encoded in a bstr type. If there are no protected attributes, a zero-length byte string is used.

sign_protected: The serialized protected attributes from the signer structure, encoded in a bstr type. If there are no protected attributes, a zero-length byte string is used. This field is omitted for the Countersignature0V2 attribute.

external_aad: The externally supplied additional authenticated data (AAD) from the application, encoded in a bstr type. If this field is not supplied, it defaults to a zero-length byte string. (See Section 4.4 of [RFC9052] for application guidance on constructing this field.)

payload: The payload to be signed, encoded in a bstr type. The payload is placed here independently of how it is transported.

other_fields: Omitted if there are only two bstr fields in the target structure. This field is an array of all bstr fields after the second. As an example, this would be an array of one element for the COSE_Sign1 structure containing the signature value.

The CDDL fragment that describes the above text is:

```
Countersign_structure = [
  context : "CounterSignature" / "CounterSignature0" /
            "CounterSignatureV2" / "CounterSignature0V2" /,
  body_protected : empty_or_serialized_map,
  ? sign_protected : empty_or_serialized_map,
  external_aad : bstr,
  payload : bstr,
  ? other_fields : [+ bstr ]
]
```

How to compute a countersignature:

1. Create a Countersign_structure and populate it with the appropriate fields.
2. Create the value ToBeSigned by encoding the Countersign_structure to a byte string, using the encoding described in Section 4.
3. Call the signature creation algorithm passing in K (the key to sign with), alg (the algorithm to sign with), and ToBeSigned (the value to sign).
4. Place the resulting signature value in the correct location. This is the "signature" field of the COSE_Countersignature structure for full countersignatures (see Section 3.1). This is the value of the Countersignature0 attribute for abbreviated countersignatures (see Section 3.2).

The steps for verifying a countersignature:

1. Create a Countersign_structure and populate it with the appropriate fields.
2. Create the value ToBeSigned by encoding the Countersign_structure to a byte string, using the encoding described in Section 4.
3. Call the signature verification algorithm passing in K (the key to verify with), alg (the algorithm used to sign with), ToBeSigned (the value to sign), and sig (the signature to be verified).

In addition to performing the signature verification, the application performs the appropriate checks to ensure that the key is correctly paired with the signing identity and that the signing identity is authorized before performing actions.

4. CBOR Encoding Restrictions

The deterministic encoding rules are defined in Section 4.2 of [RFC8949]. These rules are further narrowed in Section 9 of [RFC9052]. The narrowed deterministic encoding rules **MUST** be used to ensure that all implementations generate the same byte string for the "to be signed" value.

5. IANA Considerations

The registries and registrations listed below were created during the processing of [RFC8152]. The majority of the actions are to update the references to point to this document.

5.1. CBOR Tags Registry

IANA created a registry titled "CBOR Tags" registry as part of processing RFC 7049, which was subsequently replaced by [RFC8949].

IANA has assigned a new tag for the CounterSignature type in the "CBOR Tags" registry.

Tag: 19
Data Item: COSE_Countersignature
Semantics: COSE standalone V2 countersignature
Reference: RFC 9338

5.2. COSE Header Parameters Registry

IANA created a registry titled "COSE Header Parameters" as part of processing [RFC8152].

IANA has registered the Countersignature version 2 (label 11) and Countersignature0 version 2 (label 12) in the "COSE Header Parameters" registry. For these entries, the "Value Type" and "Description" are shown in Table 1, the "Value Registry" is blank, and the "Reference" is "RFC 9338".

Name	Label	Value Type	Description
Countersignature version 2	11	COSE_Countersignature / [+ COSE_Countersignature]	V2 countersignature attribute
Countersignature0 version 2	12	COSE_Countersignature0	V2 Abbreviated Countersignature

Table 2: New Common Header Parameters

IANA has modified the existing "Description" field for "counter signature" (7) and "CounterSignature0" (9) to include the words "(Deprecated by RFC 9338)".

6. Security Considerations

Please review the Security Considerations section in [RFC9052]; these considerations apply to this document as well, especially the need for implementations to protect private key material.

When either COSE_Encrypt or COSE_Mac is used and more than two parties share the key, data origin authentication is not provided. Any party that knows the message-authentication key can compute a valid authentication tag; therefore, the contents could originate from any one of the parties that share the key.

Countersignatures of COSE_Encrypt and COSE_Mac with short authentication tags do not provide the security properties associated with the same algorithm used in COSE_Sign. To provide 128-bit security against collision attacks, the tag length **MUST** be at least 256 bits. A countersignature of a COSE_Mac with AES-MAC (using a 128-bit key or larger) provides at most 64 bits of integrity protection. Similarly, a countersignature of a COSE_Encrypt with AES-CCM-16-64-128 provides at most 32 bits of integrity protection.

7. References

7.1. Normative References

[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/info/rfc2119>.

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

[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/info/rfc9052>.

7.2. Informative References

[CBORDIAG] Bormann, C., "CBOR diagnostic utilities", commit 1952a04, September 2022, <https://github.com/cabo/cbor-diag>.

[GROUP-OSCORE] Tiloca, M., Selander, G., Palombini, F., Mattsson, J., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-16, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-16>.

[RFC4998] Gondrom, T., Brandner, R., and U. Pordesch, "Evidence Record Syntax (ERS)", RFC 4998, DOI 10.17487/RFC4998, August 2007, <https://www.rfc-editor.org/info/rfc4998>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[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/info/rfc8610>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[STD90] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017.

<https://www.rfc-editor.org/info/std90>

Appendix A. Examples

This appendix includes a set of examples that show the different features and message types that have been defined in this document. To make the examples easier to read, they are presented using the extended CBOR diagnostic notation (defined in [RFC8610]) rather than as a binary dump.

The examples are presented using the CBOR diagnostic notation. A Ruby-based tool exists [CBORDIAG] that can convert between the diagnostic notation and binary. The referenced webpage includes installation instructions.

The diagnostic notation can be converted into binary files using the following command line:

```
diag2cbor.rb < inputfile > outputfile
```

The examples can be extracted from the XML version of this document via an XPath expression, as all of the sourcecode is tagged with the attribute 'type="cbor-diag"'. (Depending on the XPath evaluator one is using, it may be necessary to deal with > as an entity.)

```
//sourcecode[@type='cbor-diag']/text()
```

This appendix uses the following terms:

AES-GCM: AES Galois/Counter Mode

CEK: content-encryption key

ECDH: Elliptic Curve Diffie-Hellman

ECDH-ES: Elliptic Curve Diffie-Hellman Ephemeral Static

ECDSA: Elliptic Curve Digital Signature Algorithm

EdDSA: Edwards-curve Digital Signature Algorithm

HKDF: HMAC-based Key Derivation Function

HMAC: Hashed Message Authentication Code

A.1. Examples of Signed Messages

A.1.1. Countersignature

This example uses the following:

Signature Algorithm: ECDSA with SHA-256, Curve P-256

The same header parameters are used for both the signature and the countersignature.

The size of the binary file is 180 bytes.

```
98(
  [
    / protected / h'',
    / unprotected / {
      / countersign / 11:[
        / protected  h'a10126' / << {
            / alg / 1:-7 / ECDSA 256 /
          } >>,
        / unprotected / {
          / kid / 4: '11'
        },
        / signature / h'5ac05e289d5d0e1b0a7f048a5d2b643813ded50bc9e4
9220f4f7278f85f19d4a77d655c9d3b51e805a74b099e1e085aacd97fc29d72f887e
8802bb6650cceb2c'
      ]
    },
    / payload / 'This is the content.',
    / signatures / [
      [
        / protected h'a10126' / << {
            / alg / 1:-7 / ECDSA 256 /
          } >>,
        / unprotected / {
          / kid / 4: '11'
        },
        / signature / h'e2aeafd40d69d19dfe6e52077c5d7ff4e408282cbefb
5d06cbf414af2e19d982ac45ac98b8544c908b4507de1e90b717c3d34816fe926a2b
98f53afd2fa0f30a'
      ]
    ]
  ]
)
```

A.2. Examples of Signed1 Messages

A.2.1. Countersignature

This example uses the following:

Signature Algorithm: ECDSA with SHA-256, Curve P-256

Countersignature Algorithm: ECDSA with SHA-512, Curve P-521

The size of the binary file is 275 bytes.

```
18(
  [
    / protected h'A201260300' / << {
      / alg / 1:-7, / ECDSA 256 /
      / ctyp / 3:0
    } >>,
    / unprotected / {
      / kid / 4: '11',
      / countersign / 11: [
        / protected h'A1013823' / << {
          / alg / 1:-36 / ECDSA 512 /
        } >>,
        / unprotected / {
          / kid / 4: 'bilbo.baggins@hobbiton.example'
        },
        / signature / h'01B1291B0E60A79C459A4A9184A0D393E034B34AF069
A1CCA34F5A913AFFFF698002295FA9F8FCBFB6FDFF59132FC0C406E98754A98F1FBF
E81C03095F481856BC470170227206FA5BEE3C0431C56A66824E7AAF692985952E31
271434B2BA2E47A335C658B5E995AEB5D63CF2D0CED367D3E4CC8FFFD53B70D115BA
A9E86961FBD1A5CF'
      ]
    },
    / payload / 'This is the content.',
    / signature / h'BB587D6B15F47BFD54D2CBFCECEF75451E92B08A514BD439
FA3AA65C6AC92DF0D7328C4A47529B32ADD3DD1B4E940071C021E9A8F2641F1D8E3B
053DDD65AE52'
  ]
)
```

A.3. Examples of Enveloped Messages

A.3.1. Countersignature on Encrypted Content

This example uses the following:

CEK: AES-GCM with 128-bit key

Recipient Class: ECDH Ephemeral-Static, Curve P-256

Countersignature Algorithm: ECDSA with SHA-512, Curve P-521

The size of the binary file is 326 bytes.

```
96(
  [
    / protected h'a10101' / << {
        / alg / 1:1 / AES-GCM 128 /
      } >>,
    / unprotected / {
      / iv / 5:h'c9cf4df2fe6c632bf7886413',
      / countersign / 11:[
        / protected h'a1013823' / << {
            / alg / 1:-36 / ES512 /
          } >>,
        / unprotected / {
          / kid / 4: 'bilbo.baggins@hobbiton.example'
        },
        / signature / h'00929663c8789bb28177ae28467e66377da12302d7f9
594d2999afa5dfa531294f8896f2b6cdf1740014f4c7f1a358e3a6cf57f4ed6fb02f
cf8f7aa989f5dfd07f0700a3a7d8f3c604ba70fa9411bd10c2591b483e1d2c31de00
3183e434d8fba18f17a4c7e3dfa003ac1cf3d30d44d2533c4989d3ac38c38b71481c
c3430c9d65e7ddff'
      ]
    },
    / ciphertext / h'7adbe2709ca818fb415f1e5df66f4e1a51053ba6d65a1a0
c52a357da7a644b8070a151b0',
    / recipients / [
      [
        / protected h'a1013818' / << {
            / alg / 1:-25 / ECDH-ES + HKDF-256 /
          } >>,
        / unprotected / {
          / ephemeral / -1:{
            / kty / 1:2,
            / crv / -1:1,
            / x / -2:h'98f50a4ff6c05861c8860d13a638ea56c3f5ad7590bbf
bf054e1c7b4d91d6280',
            / y / -3:true
          },
          / kid / 4: 'meriadoc.brandybuck@buckland.example'
        },
        / ciphertext / h''
      ]
    ]
  ]
)
```

A.4. Examples of Encrypted Messages

A.4.1. Countersignature on Encrypted Content

This example uses the following:

CEK: AES-GCM with 128-bit key

Countersignature Algorithm: EdDSA with Curve Ed25519

The size of the binary file is 136 bytes.

```
16(
  [
    / protected h'A10101' / << {
      / alg / 1:1 / AES-GCM 128 /
    } >>,
    / unprotected / {
      / iv / 5: h'02D1F7E6F26C43D4868D87CE',
      / countersign / 11: [
        / protected h'A10127' / << {
          / alg / 1:-8 / EdDSA with Ed25519 /
        } >>,
        / unprotected / {
          / kid / 4: '11'
        },
        / signature / h'E10439154CC75C7A3A5391491F88651E0292FD0FE0E0
2CF740547EAF6677B4A4040B8ECA16DB592881262F77B14C1A086C02268B17171CA1
6BE4B8595F8C0A08'
      ]
    },
    / ciphertext / h'60973A94BB2898009EE52ECFD9AB1DD25867374B162E2C0
3568B41F57C3CC16F9166250A'
  ]
)
```

A.5. Examples of MACed Messages

A.5.1. Countersignature on MAC Content

This example uses the following:

MAC Algorithm: HMAC/SHA-256 with 256-bit key

Countersignature Algorithm: EdDSA with Curve Ed25519

The size of the binary file is 159 bytes.

```
97(
  [
    / protected h'A10105' / << {
      / alg / 1:5 / HS256 /
    } >>,
    / unprotected / {
      / countersign / 11: [
        / protected h'A10127' / << {
          / alg / 1:-8 / EdDSA /
        } >>,
        / unprotected / {
          / kid / 4: '11'
        },
        / signature / h'602566F4A311DC860740D2DF54D4864555E85BC036EA
5A6CF7905B96E499C5F66B01C4997F6A20C37C37543ADEA1D705347D38A5B13594B2
9583DD741F455101'
      ]
    },
    / payload / 'This is the content.',
    / tag / h'2BDCC89F058216B8A208DDC6D8B54AA91F48BD63484986565105C9
AD5A6682F6',
    / recipients / [
      [
        / protected / h'',
        / unprotected / {
          / alg / 1: -6, / direct /
          / kid / 4: 'our-secret'
        },
        / ciphertext / h''
      ]
    ]
  ]
)
```

A.6. Examples of MAC0 Messages

A.6.1. Countersignature on MAC0 Content

This example uses the following:

MAC Algorithm: HMAC/SHA-256 with 256-bit key

Countersignature Algorithm: EdDSA with Curve Ed25519

The size of the binary file is 159 bytes.

```
17(
  [
    / protected h'A10105' / << {
      / alg / 1:5 / HS256 /
    } >>,
    / unprotected / {
      / countersign / 11: [
        / protected h'A10127' / << {
          / alg / 1:-8 / EdDSA /
        } >>,
        / unprotected / {
          / kid / 4: '11'
        },
        / signature / h'968A315DF6B4F26362E11F4CFD2F2F4E76232F39657B
F1598837FF9332CDDD7581E248116549451F81EF823DA5974F885B681D3D6E38FC41
42D8F8E9E7DC8F0D'
      ]
    },
    / payload / 'This is the content.',
    / tag / h'A1A848D3471F9D61EE49018D244C824772F223AD4F935293F1789F
C3A08D8C58'
  ]
)
```

Acknowledgments

This document is a product of the COSE Working Group of the IETF.

The initial draft version of the specification was based to some degree on the outputs of the JOSE and S/MIME Working Groups.

Jim Schaad passed on 3 October 2020. This document is primarily his work. Russ Housley served as the document editor after Jim's untimely death, mostly helping with the approval and publication processes. Jim deserves all credit for the technical content.

Jim Schaad and Jonathan Hammell provided the examples in Appendix A.

The reviews by Carsten Bormann, Ben Kaduk, and Elwyn Davies greatly improved the clarity of the document.

Author's Address

Jim Schaad
August Cellars
United States of America