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 INSERT the number assigned to [I-D.ietf-cose-rfc8152bis-struct].

Contributing to this document

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

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/cose-wg/countersign. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the COSE mailing list.

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 23 February 2023.
Table of Contents

1. Introduction
   1.1. Requirements Terminology
   1.2. CBOR Grammar
   1.3. Document Terminology
2. Countersignature Header Parameters
3. Version 2 Countersignatures
   3.1. Full Countersignatures
   3.2. Abbreviated Countersignatures
   3.3. Signing and Verification Process
4. CBOR Encoding Restrictions
5. IANA Considerations
   5.1. CBOR Tag Assignment
   5.2. COSE Header Parameters Registry
6. Security Considerations
7. Implementation Status
   7.1. Author's Versions
   7.2. COSE Testing Library
8. References
   8.1. Normative References
   8.2. Informative References
Appendix A. Examples
   A.1. Use of Early Code Points
   A.2. Examples of Signed Messages
      A.2.1. Countersignature
   A.3. Examples of Signed1 Messages
      A.3.1. Countersignature
   A.4. Examples of Enveloped Messages
      A.4.1. Countersignature on Encrypted Content
   A.5. Examples of Encrypted Messages
      A.5.1. Countersignature on Encrypted Content
   A.6. Examples of MACed Messages
      A.6.1. Countersignature on MAC Content
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)" [STD94]. CBOR extended the data model of the JavaScript Object Notation (JSON) [STD99] 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.

During the process of advancing COSE to Internet Standard, it was noticed that the description of the security properties of countersignatures was incorrect for the COSE_Sign1 structure in Section 4.5 of [RFC8152]. To remedy this situation, the working group decided to remove all of the countersignature text from [I-D.ietf-cose-rfc8152bis-struct], 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 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 cryptographic computed value.

The new algorithm defined in this document is designed to produce the same countersignature value in those cases where the cryptographic computed 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 used 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 is uses the CBOR Data Definition Language (CDDL) [RFC8610].

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

```xml
//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.

```cddl
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".

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

"Context" is used throughout the document to represent information that is not part of the COSE message. Information which is part of the context can come from several different sources including: protocol interactions, associated key structures, and application configuration. The context to use can be implicit, identified using the "kid context" header parameter defined in [RFC8613], or identified by a protocol-specific identifier. Context should generally be included in the cryptographic construction; for more details see Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct].

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 label and the type of the value.

The set of header parameters defined in this section are:

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: 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.

<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>counter signature version 2</td>
<td>TBD10</td>
<td>COSE_Countersignature / [+ COSE_Countersignature]</td>
<td></td>
<td>V2 counter signature attribute</td>
</tr>
<tr>
<td>counter signature 0 version 2</td>
<td>TBD11</td>
<td>COSE_Countersignature0</td>
<td></td>
<td>Abbreviated Counter signature version 2</td>
</tr>
</tbody>
</table>

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; header parameters required in specific maps are discussed above.

\[
\text{CountersignatureV2\_header = (}
\text{  TBD10 => COSE\_Countersignature / [+COSE\_Countersignature]}
\text{)}
\]

\[
\text{Countersignature0V2\_header = (}
\text{  TBD11 => COSE\_Countersignature0}
\text{)}
\]

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 match 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 [I-D.ietf-core-oscore-groupcomm].

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 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 will appear to result in a successful decryption (the tag check success), but which produce two completely different plaintexts. 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 is 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 TBD0. 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 = #6.9999(COSE_Countersignature)
COSE_Countersignature = COSE_Signature
```

The details of the fields of a countersignature can be found in Section 4.1 of [I-D.ietf-cose-rfc8152bis-struct].

An example of a countersignature on a signature can be found in Appendix A.2.1. An example of a countersignature in an encryption object can be found in Appendix A.4.1.

It should be noted that only a signature algorithm with appendix (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-struct]) 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. The attribute is defined below.

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 it's cryptographic functions finalized before the 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 signatures using the COSE_Countersignature structure when other_fields is absent.

"CounterSignature0" for signatures using the COSE_Countersignature0 structure when other_fields is absent.

"CounterSignatureV2" for signatures using the COSE_Countersignature structure when other_fields is present.

"CounterSignature0V2" for signatures 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 data from the application encoded in a bstr type. If this field is not supplied, it defaults to a zero-length byte string. (See Section 4.3 of [I-D.ietf-cose-rfc8152bis-struct] for application guidance on constructing this field.)

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

other_fields: If there are only two bstr fields in the target structure, this field is omitted. The 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 are:

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 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 defined in Section 4.2 of [RFC8949]. These rules are further narrowed in Section 9 of [I-D.ietf-cose-rfc8152bis-struct]. 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 processing of RFC 8152 [RFC8152]. The majority of the actions are to update the references to point to this document.

5.1. CBOR Tag Assignment

IANA is requested to assign a new tag for the CounterSignature type in the "CBOR Tags" registry.

*Tag: TBD0

*Data Item: COSE_Countersignature

*Semantics: COSE standalone V2 countersignature

*Reference: [[this document]]

5.2. COSE Header Parameters Registry

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

IANA is requested to register the following new items in the registry. For these entries, the Value Registry column will be blank and the Reference column will be [[This Document]].

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Countersignature</td>
<td>TBD10</td>
<td>COSE_Countersignature / [+ COSE_Countersignature</td>
<td>V2 countersignature attribute</td>
</tr>
<tr>
<td>version 2</td>
<td></td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>Countersignature0</td>
<td>TBD11</td>
<td>COSE_Countersignature0</td>
<td>V2 Abbreviated Countersignature</td>
</tr>
<tr>
<td>version 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: New Common Header Parameters

IANA is requested to modify the Description field for "counter signature" and "CounterSignature0" to include the words "(Deprecated by [[This Document]])".
6. Security Considerations

Please review the Security Consideration in [I-D.ietf-cose-rfc8152bis-struct]; they apply to this document as well, especially the need for implementations to protect private key material.

When either COSE_Encrypt and 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. Implementation Status

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

7.1. Author's Versions

There are three different implementations that have been created by the author of the document both to create the examples that are
included in the document and to validate the structures and methodology used in the design of COSE.

*Implementation Location: https://github.com/cose-wg

*Primary Maintainer: Jim Schaad

*Languages: There are three different languages that are currently supported: Java and C#.

*Cryptography: The Java and C# libraries use Bouncy Castle to provide the required cryptography.

*Coverage: Both implementations can produce and consume both the old and new countersignatures.

*Testing: All of the examples in the example library are generated by the C# library and then validated using the Java and C libraries. Both libraries have tests to allow for the creating of the same messages that are in the example library followed by validating them. These are not compared against the example library. The Java and C# libraries have unit testing included. Not all of the MUST statements in the document have been implemented as part of the libraries. One such statement is the requirement that unique labels be present.

*Licensing: Revised BSD License

7.2. COSE Testing Library

*Implementation Location: https://github.com/cose-wg/Examples

*Primary Maintainer: Jim Schaad

*Description: A set of tests for the COSE library is provided as part of the implementation effort. Both success and fail tests have been provided. All of the examples in this document are part of this example set.

*Coverage: An attempt has been made to have test cases for every message type and algorithm in the document. Currently examples dealing with countersignatures, and ECDH with Curve25519 and Goldilocks are missing.

*Licensing: Public Domain

8. References

8.1. Normative References
8.2. Informative References


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's diagnostic notation. A Ruby-based tool exists that can convert between the diagnostic notation and binary. This tool can be installed with the command line:

gem install cbor-diag

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
A.1. Use of Early Code Points

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

The examples in this Appendix use code points proposed for early allocation by IANA. When IANA makes the allocation, these examples will be updated as needed.

A.2. Examples of Signed Messages

A.2.1. Countersignature

This example uses the following:

*Signature Algorithm: ECDSA w/ SHA-256, Curve P-256

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

Size of binary file is 180 bytes
A.3. Examples of Signed1 Messages

A.3.1. Countersignature

This example uses the following:

*Signature Algorithm: ECDSA w/ SHA-256, Curve P-256

*Countersignature Algorithm: ECDSA w/ SHA-512, Curve P-521

Size of binary file is 275 bytes
A.4. Examples of Enveloped Messages

A.4.1. Countersignature on Encrypted Content

This example uses the following:

*CEK: AES-GCM w/ 128-bit key

*Recipient class: ECDH Ephemeral-Static, Curve P-256

*Countersignature Algorithm: ECDSA w/ SHA-512, Curve P-521

Size of binary file is 326 bytes
A.5. Examples of Encrypted Messages

A.5.1. Countersignature on Encrypted Content

This example uses the following:

*CEK: AES-GCM w/ 128-bit key

*Countersignature algorithm: EdDSA with Curve Ed25519

Size of 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'E10439154CC75C7A3A5391491F88651E0292FD0FE0E02CF740547EAF6677B4A4040B8ECA16DB592881262F77B14C1A086C02268B17171CA16BE4B8595F8C0A08'  
         }  
      },  
      / ciphertext / h'60973A94BB2898009EE52ECDF9AB1DD25867374B162E2C0356B41F57C3CC16F9166250A'  
   ]
)

A.6. Examples of MACed Messages

A.6.1. Countersignature on MAC Content

This example uses the following:

*MAC algorithm: HMAC/SHA-256 with 256-bit key

*Countersignature algorithm: EdDSA with Curve Ed25519

Size of binary file is 159 bytes
A.7. Examples of MAC0 Messages

A.7.1. Countersignature on MAC0 Content

This example uses the following:

*MAC algorithm: HMAC/SHA-256 with 256-bit key
*Countersignature algorithm: EdDSA with Curve Ed25519

Size of binary file is 159 bytes
Acknowledgments

This document is a product of the COSE working group of the IETF.

The initial 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 the Appendix.

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

{{{{ RFC EDITOR: Please remove Russ Housley as an author of this document at publication. Jim Schaad should be listed as the sole author. }}}}}
Authors' Addresses

Jim Schaad
August Cellars
United States of America

Email: ietf@augustcellars.com

Russ Housley (editor)
Vigil Security, LLC
United States of America

Email: housley@vigilsec.com