Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 18, 2020

T. Looker, Ed.
Mattr

January 15, 2020

JSON Web Message
draft-looker-jwm-01

Abstract

JSON Web Message (JWM) is a flexible way to encode application-level messages in JSON for transfer over a variety of transport protocols. JWMs use JSON Web Encryption (JWE) to protect integrity, achieve confidentiality, and achieve repudiable authentication; alternatively or in addition, they use JSON Web Signatures (JWS) to associate messages with a non-repudiable digital signature.

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 July 18, 2020.

Copyright Notice

Copyright (c) 2020 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 Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

JSON Web Message (JWM) is a flexible way to encode application-level messages in JSON for transfer over a variety of transport protocols. JWMs use JSON Web Encryption (JWE) to protect integrity, achieve confidentiality, and achieve repudiable authentication; alternatively or in addition, they use JSON Web Signatures (JWS) to associate messages with a non-repudiable digital signature. JWMs are inspired by JWTs [RFC7519]; more details about this relationship are documented in Section 2.1.

1.1. Notational Conventions

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 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Terminology

The terms "JSON Web Signature (JWS)", "Base64url Encoding", "Header Parameter", "JOSE Header", "JWS Payload", "JWS Signature", "JWS Compact Serialization" and "JWS JSON Serialization" are defined by the JWS specification [RFC7515].

The terms "JSON Web Encryption (JWE)", "JWE Compact Serialization" and "JWE JSON Serialization" are defined by the JWE specification [RFC7516].

The terms "StringOrURI", "NumericDate" are defined by the JWT specification [RFC7519].

The following terms are defined by this specification:

JSON Web Message A JWM Attribute Set, encoded in a JWS and/or JWE, enabling it to be digitally signed and/or encrypted.

JWM Attribute Set A JSON object of attributes conveyed by the JWM.

Attribute A piece of information conveyed in a message, sent from a sender intended for processing by one or more recipients. An attribute is represented in a JWM Attribute Set as a name/value pair consisting of an Attribute Name and an Attribute Value.

Attribute Name The name portion of an attribute representation. An attribute name is always a string.

Attribute Value The value portion of an attribute representation. An attribute value can be any JSON value.

Nested JWM A JWM in which nested signing and/or encryption is employed. In Nested JWMs, a JWM is used as the payload or plaintext value of an enclosing JWS or JWE structure, respectively.

2. JSON Web Message (JWM) Overview

2.1. Relationship to JWT

JWMs conceptually share parallels to JWTs:

- A JWM contains a JSON object comprised of attributes known as a JWM Attribute Set, where the attributes featured in the set can be public, private or registered in an IANA registry. This is conceptually parallel to claims in JWTs, the JWT Claim Set, and the JWT claims IANA registry.

- A JWM leverages JSON Web Signature (JWS) and or JSON Web Encryption (JWE) to achieve digital signing, integrity protection and or confidentiality via encryption for the JWM attribute set in similar ways to JWT for the JWT claim set.

JWMs also deviate from JWTs in important ways that prevent a converged specification.

- JWM and JWT have different intents. A JWM is about a sender creating a message composed of attributes, where the message is destined for a recipient or recipients. Whereas a JWT is about an issuer expressing claims about a subject to an audience.

- The primary usage of JWTs centers around creating tokens that are digitally signed or integrity protected through a Message Authentication Code (MAC) by leveraging JWS. Encrypted JWTs via JWE are less common and as defined in Section 8 of [RFC7519] they are optional to implement. Whereas JWMs require both JWS and JWE implementations.

- Because JWTs must be compact and URL-safe, they require compact serialization for both JWS and JWE representations. This means JWTs can feature only a single digital signature, and/or encrypt for only a single recipient. In contrast, JWMs support multiple

digital signatures, and encryption for multiple recipients. They achieve this using the JSON-based serialization of JWS and JWE.

- JWTs support an insecure format as defined in Section 6 of [RFC7519], JWMs do not allow this format due to the vulnerabilities it introduces.

In the spirit of specification re-use and promoting ease of understanding, this specification's structure is inspired by [RFC7519].

2.2. Example Signed JWM

The following example JOSE Header declares that the object is a JWM, and the JWM is a JWS that has been digitally signed using ECDSA with the curve p-256 and SHA-256 as the hashing algorithm ("ES256"). The signer of the JWM has indicated that the key used to sign the JWM is identified by "Ef1sFuyOozYm3CEY4iCdwqxiSyXZ5Br-eUDdQXk6jaQ" (with line breaks for display purposes only):

```
{"typ":"JWM",
 "kid":"Ef1sFuyOozYm3CEY4iCdwqxiSyXZ5Br-eUDdQXk6jaQ",
 "alg":"ES256"}
```

Base64url encoding the octets of the UTF-8 representation of the JOSE Header yields this encoded JOSE Header value (with line breaks for display purposes only):

```
eyJ0eXAiOiJKV00iLCJraWQiOiJFZjFzRnV5T296WW0zQ0VZNGlDZHdxeGlTeVhaNUJyL
WVVRGRRWGs2amFRIiwiYWxnIjoiRVMyNTYifQ
```

The following is an example of a JWM Attributes Set (with line breaks for display purposes only):

```
{"id":"urn:uuid:ef5a7369-f0b9-4143-a49d-2b9c7ee51117",
 "type":"hello-world-message-type",
 "from":"urn:uuid:8abdf5fb-621e-4cf5-a595-071bc2c91d82",
 "expires_time":1516239022,
 "created_time":1516269022,
 "body":{"message": "Hello world!"}}
```

Base64url encoding the octets of the UTF-8 the JWS Payload yields this encoded JWS Payload (with line breaks for display purposes only):

```
eyJpZCI6InVybjp1dWlkOmVmNWE3MzY5LWYwYjktNDE0My1hNDlkLTJiOWM3ZWU1MTExN
yIsInR5cGUiOiJoZWxsby13b3JsZC1tZXNzYWdlLXR5cGUiLCJmcm9tIjoidXJuOnV1aW
Q6OGFiZGY1ZmItNjIxZS00Y2Y1LWE1OTUtMDcxYmMyYzkxZDgyIiwiZXhwaXJ5IjoxNTE
```

2MjM5MDIyLCJ0aW1lX3N0YW1wIjoxNTE2MjY5MDIyLCJib2R5Ijp7Im1lc3NhZ2UiOiJI
ZWxsbyB3b3JsZCEifX0

Computing the signature of the encoded JOSE Header and encoded JWS
Payload with the ECDSA with the curve p-256 and SHA-256 as the
hashing algorithm and base64url encoding the value in the manner
specified in [RFC7515] yields this encoded JWS Signature:

UDE7NsEJyhiewrX2_Z9OdIdB2ZREauoPoUAKdEmW72d8H_ivkjC1p7G16WHunBMq1zFka
nINTil3-H1FlhbzsQ

Compact Serialization (with line breaks for display purposes only):

eyJ0eXAiOiJKV00iLCJraWQiOiJFZjFzRnV5T296WW0zQ0VZNGlDZHdxeGlTeVhaNUJyL
WVVRGRRWGs2amFRIiwiYWxnIjoiRVMyNTYifQ
.
eyJpZCI6InVybjp1dWlkOmVmNWE3MzY5LWYwYjktNDE0My1hNDlkLTJiOWM3ZWU1MTExN
yIsInR5cGUiOiJoZWxsby13b3JsZC1tZXNzYWdlLXR5cGUiLCJmcm9tIjoidXJuOnV1aW
Q6OGFiZGY1ZmItNjIxZS00Y2Y1LWE1OTUtMDcxYmMyYzkxZDgyIiwiZXhwaXJ5IjoxNTE
2MjM5MDIyLCJ0aW1lX3N0YW1wIjoxNTE2MjY5MDIyLCJib2R5Ijp7Im1lc3NhZ2UiOiJI
ZWxsbyB3b3JsZCEifX0
.
UDE7NsEJyhiewrX2_Z9OdIdB2ZREauoPoUAKdEmW72d8H_ivkjC1p7G16WHunBMq1zFka
nINTil3-H1FlhbzsQ

JSON Serialization: (with line breaks for display purposes only):

    {
      "payload": "eyJpZCI6InVybjp1dWlkOmVmNWE3MzY5LWYwYjktNDE0My1hNDl
      kLTJiOWM3ZWU1MTExNyIsInR5cGUiOiJoZWxsby13b3JsZC1tZXNzYWdlLXR5cG
      UiLCJmcm9tIjoidXJuOnV1aWQ6OGFiZGY1ZmItNjIxZS00Y2Y1LWE1OTUtMDcxY
      mMyYzkxZDgyIiwiZXhwaXJ5IjoxNTE2MjM5MDIyLCJ0aW1lX3N0YW1wIjoxNTE2
      MjY5MDIyLCJib2R5Ijp7Im1lc3NhZ2UiOiJIZWxsbyB3b3JsZCEifX0",
      "signatures": [
        {
          "protected": "eyJ0eXAiOiJKV00iLCJraWQiOiJFZjFzRnV5T296WW0zQ
          0VZNGlDZHdxeGlTeVhaNUJyLWVVRGRRWGs2amFRIiwiYWxnIjoiRVMyNTYi
          fQ",
          "signature": "rwhHoGJZRyLliF2jPqGXMddBLWlJls4XqSO21GH2itlwh
          3d3Zb2jAtqA93s9Lb6ktXoxqHxNy4Lbirtr3pCHQA"
        }
      ]
    }

2.3. Example Encrypted JWM

The following example JOSE Header declares that the object is a JWM, and the JWM is a JWE that has been encrypted using AES in Galois/ Counter Mode (GCM) with 256-bit for content encryption (with line breaks for display purposes only):

```
{
 "typ":"JWM",
 "enc":"A256GCM"
}
```

The following JOSE Header declares that the sender has used Elliptic Curve Diffie-Hellman Ephemeral Static (ECDH-ES+A256KW) for key agreement with AES based key wrapping to encrypt the content encryption key (CEK). The keys required to perform the opposite Diffie-Helman are identified by the key id of "PGoXzs0NWaR_meKgTZLbEuDoSVTaFuyrbWI7V9dpjCg" along with the ephemeral public key declared by the JSON object "epk". The encrypted CEK is represented by the "encrypted_key" field (with line breaks for display purposes only):

```
{
"kid": "PGoXzs0NWaR_meKgTZLbEuDoSVTaFuyrbWI7V9dpjCg",
"alg": "ECDH-ES+A256KW",
"epk": {
  "kty": "EC",
  "crv": "P-256",
  "x": "-Nh7ShRB_xaCBZRdIiVCul3SoR0Yw4TGEQqqGij1vJs",
  "y": "9tLx81PMfQkrOw8yuI2YwI0o7MtNzaCGfCBbZBW5YrM"
}
}
```

Because the JWE features a single recipient, the two JOSE headers are combined into a single JOSE header, that can be represented as the following.

```
{
  "typ": "JWM",
  "enc": "A256GCM",
  "kid": "PGoXzs0NWaR_meKgTZLbEuDoSVTaFuyrbWI7V9dpjCg",
  "alg": "ECDH-ES+A256KW",
  "epk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "-Nh7ShRB_xaCBZRdIiVCul3SoR0Yw4TGEQqqGij1vJs",
    "y": "9tLx81PMfQkrOw8yuI2YwI0o7MtNzaCGfCBbZBW5YrM"
  }
}
```

The following is an example of a JWM Attributes Set (with line breaks for display purposes only):

```
{"id":"urn:uuid:ef5a7369-f0b9-4143-a49d-2b9c7ee51117",
 "type":"hello-world-message-type",
 "from":"urn:uuid:8abdf5fb-621e-4cf5-a595-071bc2c91d82",
 "expires_time":1516239022,
 "created_time":1516269022,
 "body":{"message": "Hello world!"}}
```

Encrypting the above plaintext for the recipient yields the following ciphertext in base64url form.

```
awEW6ssGMbQxmvv4FPf0smom4QvPNrgLaxFiMMRXmUTgcs6mLcSJDbUhwLfGfnEeu2a0b
cGLRt7tTuQij5RBIe6sflhIgOjpr3VAHdZBYJbF Jg9dCMW_hVk0iLytmFV5BhvqXUXDA
ckwwTU41PcS2_qO5uqdIe24teP8Bd_IbVeVnaUwrEEBGJvxYDTefdZ4gryrzKFsLLBD5F
r9TsCFEddg0RL xaXFGX1YT8Jm6Ahm-jd6Ol9qIpWx-
8PMaFcZl7h4sPiAGVPiaaCyzTsMy8KW0Nmz3cEFqjEm4Ipc
```

Composing this information into a valid JWE leads to the following possible expressions

Compact Serialization (with line breaks for display purposes only):

```
eyJ0eXAiOiJKV00iLCJlbmMiOiJBMjU2R0NNIiwia2lkIjoiUEdvWHpzME5XYVJfbWVLZ
1RaTGJFdURvU1ZUYUZ1eXJiV0k3VjlkcGpDZyIsImFsZyI6I kVDREgtRVMrQTI1NktXI
iwiZXBrIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiZGdMdy1wOG5kZ0xRSm
hZeWhUaGhpVDRhbmJlRjhaak1MYXRxR 2dXVGxHSSIsInkiOiJ3MkNfcjUzekdUdVZscD
hQVndFZjViWWI0TWo4bXVjNTVtMHh6VkVMN1o0In19
.
rAiydPRY_cciOmaQ-tnNiacHWn2Z2GqDgf0FcG4nK2L_KsPd1V1OSA
.
EIY6u2ahL0MI28ah
.
```

```
awEW6ssGMbQxmvv4FPf0smom4QvPNrgLaxFiMMRXmUTgcs6mLcSJDbUhwLfGfnEeu2a0b
cGLRt7tTuQij5RBIe6sflhIgOjpr3VAHdZBYJbF Jg9dCMW_hVk0iLytmFV5BhvqXUXDA
ckwwTU41PcS2_qO5uqdIe24teP8Bd_IbVeVnaUwrEEBGJvxYDTefdZ4gryrzKFsLLBD5F
r9TsCFEddg0RL xaXFGX1YT8Jm6Ahm-jd6Ol9qIpWx-
8PMaFcZl7h4sPiAGVPiaaCyzTsMy8KW0Nmz3cEFqjEm4Ipc
.
fp_tT_6qsQK2d9szRAwWgA
```

JSON Serialization (with line breaks for display purposes only):

```
  {
  "protected": "eyJ0eXAiOiJKV00iLCJlbmMiOiJBMjU2R0NNIiwi
  a2lkIjoiUEdvWHpzME5XYVJfbWVLZ1RaTGJFdURvU1ZUYUZ1eXJiV0
  k3VjlkcGpDZyIsImFsZyI6IkVDREgtRVMrQTI1NktXIiwiZXBrIjp7
  Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiLU5oN1NoUkJfeG
  FDQlpSZElpVkN1bDNTb1IwWXc0VEdFUXFxR2lqMXZKcyIsInkiOiI5
  dEx4ODFQTWZRa3JPdzh5dUkyWXdJMG83TXROemFDR2ZDQmJaQlc1WX
  JNIn19",
  "recipients": [
    {
      "encrypted_key": "J1Fs9JaDjOT_5481ORQWfEZmHy7OjE3p
      TNKccnK7hlqjxbPalQWWLg"
    }
  ],
  "iv": "u5kIzo0m_d2PjI4m",
  "ciphertext": "qGuFFoHy7HBmkf2BaY6eREwzEjn6O_FnRoXj2H-
  DAXo1PgQdfON-_1QbxtnT8e8z_M6Gown7s8fLtYNmIHAuixqFQnSA4
  fdMcMSi02z1MYEn2JC-1EkVbWr4TqQgFP1EyymB6XjCWDiwTYd2xpK
  oUshu8WW601HLSgFIRUG3-cK_ZSdFaoWosIgAH5EQ2ayJkRB_7dXuo
  6_AYdIzMahvPz0n1yHHBlYBuYeR58V-x85ACeCGtzL2OptPa2TmWdA
  9Bi1MK6TYGZKezc6rpCK_VRSnLXhFwa1C3T0QBes",
  "tag": "doeAoagwJe9BwKayfcduiw"
}
```

3. JWM Attributes

The JWM Attributes Set represents a JSON object whose members are the attributes conveyed by the JWM. The Attribute Names within a JWM Attributes Set MUST be unique; JWM parsers MUST either reject JWMs with duplicate Attribute Names or use a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 ("The JSON Object") of ECMAScript 5.1 [ECMAScript].

The set of attributes that a JWM must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of JWMs will require implementations to understand and process some attributes in particular ways. However,

in the absence of such requirements, all attributes that are not understood by implementations MUST be ignored.

There are three classes of JWM Attribute Names: Registered Attribute Names, Public Attribute Names and Private Attribute Names.

3.1. Registered Attribute Names

The following Attribute Names are registered in the IANA "JSON Web Message Attributes" registry established by Section 7. None of the attributes defined below are intended to be mandatory to use or implement in all cases, but rather they provide a starting point for a set of useful, interoperable attributes. Applications using JWMs should define which specific Attributes they use and when they are required or optional.

3.1.1. "id" Attribute

The "id" attribute is used to define a unique identifier for a JWM. The "id" attribute value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to another JWM. The processing of this attribute is generally application specific. The "id" attribute value is a case-sensitive string containing a StringOrURI value. Use of this attribute is OPTIONAL.

3.1.2. "type" Attribute

The "type" attribute is used to define the type of the message. The processing of this attribute is generally application specific. The "type" attribute value is a case-sensitive string containing a StringOrURI value. The "type" attribute value can be used by applications to inform the structure and content of the "message body" and indicate the presence of other JWM attributes. Use of this attribute is OPTIONAL.

3.1.3. "body" Attribute

The "body" attribute is used to define a location for application level message content. The "body" attribute value is a JSON object conforming to RFC 7159 [RFC7159]. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.4. "to" Attribute

The "to" attribute is used to define the intended recipients of the JWM. The "to" attribute value is an array of case-sensitive strings each containing a StringOrURI value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.5. "from" Attribute

The "from" attribute is used to define the sender of the JWM. The "from" attribute value is a case-sensitive string containing a StringOrURI value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.6. "thread_id" Attribute

The "thread_id" attribute is used to associate the JWM to a group of related messages often referred to as a thread. The "thread_id" attribute value is a case-sensitive string containing a StringOrURI value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.7. "created_time" Attribute

The "created_time" attribute is used to define the time in which the message was created. The "created_time" attributes value MUST be a number containing a NumericDate value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.8. "expires_time" Attribute

The "expires_time" attribute is used to define the lifespan or lifetime of the JWM. The "expires_time" attributes value MUST be a number containing a NumericDate value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.9. "reply_url" Attribute

The "reply_url" attribute is used to define a url to which a response to the message can be sent. The "reply_url" attribute value is a case-sensitive string containing a StringOrURI value. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.1.10. "reply_to" Attribute

The "reply_to" attribute is used to define who a response to the message should be sent to. The "reply_to" attribute value is a case-sensitive string containing an array of case-sensitive strings containing StringOrURI values. The processing of this attribute is generally application specific. Use of this attribute is OPTIONAL.

3.2. Public Attribute Names

Attribute Names can be defined at will by those using JWMs. However, in order to prevent collisions, any new Attribute Name should either be registered in the IANA "JSON Web Message Attributes" registry established by Section 7 or be a Public Name: a value that contains a Collision-Resistant Name. In each case, the definer of the name or value needs to take reasonable precautions to make sure they are in control of the part of the namespace they use to define the Attribute Name.

3.3. Private Attribute Names

A producer and consumer of a JWM MAY agree to use Attribute Names that are Private Names: names that are not Registered Attribute Names Section 3.1 or Public Attribute Names Section 3.2. Unlike Public Attribute Names, Private Attribute Names are subject to collision and should be used with caution.

4. JOSE Header

For a JWM object, the members of the JSON object represented by the JOSE Header describe the cryptographic operations applied to the JWM and optionally, additional properties of the JWM. Depending upon whether the JWM is a JWS or JWE, the corresponding rules for the JOSE Header values apply.

This specification further specifies the use of the following header parameters in both the cases where the JWM is a JWS and where it is a JWE.

4.1. "typ" (Type) Header Parameter

The "typ" (type) Header Parameter defined by [RFC7515] and [RFC7516] is used by JWM applications to declare the media type [IANA.MediaTypes] of this complete JWM. This is intended for use by the JWM application when values that are not JWMs could also be present in an application data structure that can contain a JWM object; the application can use this value to disambiguate among the different kinds of objects that might be present. It will typically

not be used by applications when it is already known that the object is a JWM. This parameter is ignored by JWM implementations; any processing of this parameter is performed by the JWM application. If present, it is RECOMMENDED that its value be "JWM" to indicate that this object is a JWM. While media type names are not case sensitive, it is RECOMMENDED that "JWM" always be spelled using uppercase characters for compatibility with legacy implementations. Use of this Header Parameter is OPTIONAL.

4.2. "cty" (Content Type) Header Parameter

The "cty" (content type) Header Parameter defined by [RFC7515] and [RFC7516] is used by this specification to convey structural information about the JWM.

In the normal case in which nested signing or encryption operations are not employed, the use of this Header Parameter is NOT RECOMMENDED. In the case that nested signing or encryption is employed, this Header Parameter MUST be present; in this case, the value MUST be "JWM", to indicate that a Nested JWM is carried in this JWM. While media type names are not case sensitive, it is RECOMMENDED that "JWM" always be spelled using uppercase characters for compatibility with legacy implementations.

4.3. Replicating Attributes as Header Parameters

In some applications using encrypted JWMs, it is useful to have an unencrypted representation of some attributes. This might be used, for instance, in application processing rules to determine whether and how to process the JWM before it is decrypted.

This specification allows Attributes present in the JWM Attributes Set to be replicated as JOSE Header Parameters in a JWM that is a JWE, as needed by the application. If such replicated attributes are present, the application receiving them SHOULD verify that their values are identical, unless the application defines other specific processing rules for these attributes. It is the responsibility of the application to ensure that only attributes that are safe to be transmitted in an unencrypted manner are replicated as JOSE Header Parameter values in the JWM.

5. Creating and Validating JWMs

5.1. Creating a JWM

To create a JWM, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Create a JWM Attribute Set containing the desired attributes. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding.

2. Let the Message be the octets of the UTF-8 representation of the JWM Attributes Set.

3. Create a JOSE Header containing the desired set of Header Parameters. The JWM MUST conform to either the JWS [RFC7515] or JWE [RFC7516] specification. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding. THE JOSE "typ" header must be set to "JWM".

4. Depending upon whether the JWM is a JWS or JWE, there are two cases:

o If the JWM is a JWS, create a JWS using the JWM Attribute Set as the JWS Payload; all steps specified in JWS [RFC7515] for creating a JWS MUST be followed. If the resulting JWS features only a single signature, it can optionally be formated into JWS compact serialization format allowing the message to be URL safe. If however, the resulting JWS features multiple signatures and URL safety for the message is still required, the entire JWS in JSON serialization format MUST be encoded to base64url format. Otherwise the output format for the JWS MUST be JWS JSON serialization format.

o Else, if the JWM is a JWE, create a JWE using the JWM Attribute Set as the plaintext for the JWE; all steps specified in JWE [RFC7516] for creating a JWE MUST be followed. If the resulting JWE features only a single recipient, it can optionally be formated into JWE compact serialization format allowing the message to be URL safe. If however, the resulting JWE features multiple recipients and URL safety for the message is still required, the entire JWE in JSON serialization format MUST be encoded to base64url format. Otherwise the output format for the JWE MUST be JWE JSON serialization format.

1. If a Nested JWM is desired, let the Message be the JWS or JWE, and return to Step 3, using a "cty" (content type) value of "JWM" in the new JOSE Header created in that step.

2. Otherwise, let the resulting JWM be the JWS or JWE.

5.2. Validating a JWM

When validating a JWM, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If any of the listed steps fail, then the JWM MUST be rejected; that is, treated by the application as an invalid input.

1. If the JWM is a valid base64url string containing at least one period ('.') character.

    1. Let the Encoded JOSE Header be the portion of the JWM before the first period ('.') character.

    2. Base64url decode the Encoded JOSE Header following the restriction that no line breaks, whitespace, or other additional characters have been used.

    3. Verify that the resulting octet sequence is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159 [RFC7159]; let the JOSE Header be this JSON object.

2. Else, if the JWM is a valid base64url string containing no period ('.') characters.

    1. Let the Encoded JWS or JWE be the entire base64url string.

    2. Base64url decode the Encoded JWS or JWE following the restriction that no line breaks, whitespace, or other additional characters have been used.

    3. Verify that the resulting octet sequence is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159 [RFC7159]; let the JWS or JWE be this JSON object.

3. Else, if the JWM is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159 [RFC7159]; let the JWS or JWE be this JSON object.

4. Verify that the resulting JOSE Header includes only parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.

5. Determine whether the JWM is a JWS or a JWE using any of the methods described in Section 9 of [RFC7516].

6. Depending upon whether the JWM is a JWS or JWE, there are two cases:

   * If the JWM is a JWS, follow the steps specified in [RFC7515] for validating a JWS. Let the Message be the result of base64url decoding the JWS Payload.

   * Else, if the JWM is a JWE, follow the steps specified in [RFC7516] for validating a JWE. Let the Message be the resulting plaintext.

7. If the JOSE Header contains a "cty" (content type) value of "JWM", then the Message is a JWM that was the subject of nested signing or encryption operations. In this case, return to Step 1, using the Message as the JWM.

8. Otherwise, base64url decode the Message following the restriction that no line breaks, whitespace, or other additional characters have been used.

9. Verify that the resulting octet sequence is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159 [RFC7159]; let the JWM Attributes Set be this JSON object.

Finally, note that it is an application decision which algorithms may be used in a given context. Even if a JWM can be successfully validated, unless the algorithms used in the JWM are acceptable to the application, it SHOULD reject the JWM.

5.3. String Comparison Rules

These rules are identical to those applied to JWTs outlined in Section 7.3 of [RFC7519].

6. Implementation Requirements

This section defines which algorithms and features of this specification are mandatory to implement. Applications using this specification can impose additional requirements upon implementations that they use.

Support for digitally signed JWMs using JWS is REQUIRED. Of the signature and MAC algorithms specified in JSON Web Algorithms [RFC7518], only ECDSA using the P-256 curve and SHA-256 hash algorithm ("ES256") MUST be implemented by conforming JWM implementations. It is RECOMMENDED that implementations also support ECDSA using the P-521 curve and the SHA-512 hash algorithm ("ES512")

and EdDSA using the Ed25519 curve and SHA-512 hash algorithm. Support for other algorithms and key sizes is OPTIONAL.

Support for encrypted JWMs using JWE is also REQUIRED. Of the encryption algorithms specified in [RFC7518], using Elliptic Curve Diffie-Hellman Ephemeral Static (ECDH-ES) to agree upon a key and using this key to to perform key wrapping of a Content Encryption Key ("ECDH-ES+A128KW" and "ECDH-ES+A256KW") MUST be supported. With regards to content encryption, AES in Galois/Counter Mode (GCM) with 128-bit and 256-bit keys ("A128GCM" and "A256GCM") MUST also be supported.

Usage of the "none" algorithm identifier in a JWM as defined in the JOSE Web Algorithms section 3.6 [RFC7518] MUST be considered invalid.

Support for Nested JWMs is also REQUIRED.

7. IANA Considerations

7.1. Registration Template

7.1.1. Attribute Name:

The name requested (e.g., "type"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short - that is, not to exceed 8 characters without a compelling reason to do so. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

7.1.2. Attribute Description

Brief description of the attribute (e.g., "Message Type").

7.1.3. Change Controller

For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

7.1.4. Specification Document(s)

Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

7.1.5. Initial Registry Contents

- o Attribute Name: "id"
- o Attribute Description: Message ID
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "type"
- o Attribute Description: Message Type
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "body"
- o Attribute Description: Message Body
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "to"
- o Attribute Description: Message Recipients
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "from"
- o Attribute Description: Message From
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "thread_id"
- o Attribute Description: Message Thread ID
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "created_time"
- o Attribute Description: Message Created Time
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "expires_time"
- o Attribute Description: Message Expiry Time
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "reply_url"
- o Attribute Description: Message Reply URL
- o Change Controller:
- o Specification Document(s):

- o Attribute Name: "reply_to"

- Attribute Description: Message Reply To
- Change Controller:
- Specification Document(s):

8. Media Type Registration

8.1. Registry Contents

9. Security Considerations

All of the security issues that are pertinent to any cryptographic application must be addressed by JWM/JWS/JWE/JWK agents. Among these issues are protecting the user's asymmetric private and symmetric secret keys and employing countermeasures to various attacks.

All the security considerations in the JWS specification also apply to JWM, as do the JWE security considerations when encryption is employed. In particular, Sections 10.12 ("JSON Security Considerations") and 10.13 ("Unicode Comparison Security Considerations") of [RFC7515] apply equally to the JWM Attributes Set in the same manner that they do to the JOSE Header.

9.1. Trust Decisions

The contents of a JWM cannot be relied upon in a trust decision unless its contents have been cryptographically secured and bound to the context necessary for the trust decision. In particular, the key(s) used to sign and/or encrypt the JWM will typically need to verifiably be under the control of the party identified by the associated cryptographic operation.

9.2. Signing and Encryption Order

While syntactically the signing and encryption operations for Nested JWMs may be applied in any order, if both signing and encryption are necessary, normally producers should sign the message and then encrypt the result (thus encrypting the signature). This prevents attacks in which the signature is stripped, leaving just an encrypted message, as well as providing privacy for the signer. Furthermore, signatures over encrypted text are not considered valid in many jurisdictions.

Note that potential concerns about security issues related to the order of signing and encryption operations are already addressed by the underlying JWS and JWE specifications; in particular, because JWE only supports the use of authenticated encryption algorithms, cryptographic concerns about the potential need to sign after

encryption that apply in many contexts do not apply to this specification.

10. Privacy Considerations

A JWM may contain privacy-sensitive information. When this is the case, measures MUST be taken to prevent disclosure of this information to unintended parties. One way to achieve this is to use an encrypted JWM and authenticate the recipient. Another way is to ensure that JWMs containing unencrypted privacy-sensitive information are only transmitted using protocols utilizing encryption that support endpoint authentication, such as Transport Layer Security (TLS). Omitting privacy-sensitive information from a JWM is the simplest way of minimizing privacy issues.

11. Acknowledgements

The authors of this specification would like to acknowledge the following individuals for the significant contribution of ideas and time spent reviewing this document:
Kyle Den Hartog
Daniel Hardman

12. Normative References

[ECMAScript]
Ecma International, "ECMAScript Language Specification, 5.1 Edition", ECMA Standard 262, June 2011, <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>.

[IANA.MediaTypes]
IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[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>.

Author's Address

Tobias Looker (editor)
Mattr

Email: tobias.looker@mattr.global