JSON Web Encryption (JWE)
draft-ietf-jose-json-web-encryption-33

Abstract

   JSON Web Encryption (JWE) represents encrypted content using
   JavaScript Object Notation (JSON) based data structures.
   Cryptographic algorithms and identifiers for use with this
   specification are described in the separate JSON Web Algorithms (JWA)
   specification and IANA registries defined by that specification.
   Related digital signature and MAC capabilities are described in the
   separate JSON Web Signature (JWS) specification.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

   JSON Web Encryption (JWE) represents encrypted content using
   JavaScript Object Notation (JSON) [RFC7159] based data structures.
   The JWE cryptographic mechanisms encrypt and provide integrity
   protection for an arbitrary sequence of octets.

   Two closely related serializations for JWE objects are defined. The
   JWE Compact Serialization is a compact, URL-safe representation
   intended for space constrained environments such as HTTP
   Authorization headers and URI query parameters. The JWE JSON
   Serialization represents JWE objects as JSON objects and enables the
   same content to be encrypted to multiple parties. Both share the
   same cryptographic underpinnings.

   Cryptographic algorithms and identifiers for use with this
   specification are described in the separate JSON Web Algorithms (JWA)
   [JWA] specification and IANA registries defined by that
   specification. Related digital signature and MAC capabilities are
   described in the separate JSON Web Signature (JWS) [JWS]
   specification.

   Names defined by this specification are short because a core goal is
   for the resulting representations to be compact.

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 Key
   words for use in RFCs to Indicate Requirement Levels [RFC2119]. If
   these words are used without being spelled in uppercase then they are
   to be interpreted with their normal natural language meanings.

   BASE64URL(OCTETS) denotes the base64url encoding of OCTETS, per
   Section 2.

   UTF8(STRING) denotes the octets of the UTF-8 [RFC3629] representation
   of STRING.

   ASCII(STRING) denotes the octets of the ASCII [USASCII]
   representation of STRING.

   The concatenation of two values A and B is denoted as A || B.

2. **Terminology**

   These terms defined by the JSON Web Signature (JWS) [JWS]
   specification are incorporated into this specification: "JSON Web
   Signature (JWS)", "Base64url Encoding", "Collision-Resistant Name",
   "Header Parameter", "JOSE Header", and "StringOrURI".

   These terms are defined by this specification:

   JSON Web Encryption (JWE)
      A data structure representing an encrypted and integrity protected
      message.

   Authenticated Encryption with Associated Data (AEAD)
      An AEAD algorithm is one that encrypts the Plaintext, allows
      Additional Authenticated Data to be specified, and provides an
      integrated content integrity check over the Ciphertext and
      Additional Authenticated Data. AEAD algorithms accept two inputs,
      the Plaintext and the Additional Authenticated Data value, and
      produce two outputs, the Ciphertext and the Authentication Tag
      value. AES Galois/Counter Mode (GCM) is one such algorithm.

   Plaintext
      The sequence of octets to be encrypted -- a.k.a., the message.
      The plaintext can contain an arbitrary sequence of octets.

   Ciphertext
      An encrypted representation of the Plaintext.

   Additional Authenticated Data (AAD)
      An input to an AEAD operation that is integrity protected but not
      encrypted.

   Authentication Tag
      An output of an AEAD operation that ensures the integrity of the
      Ciphertext and the Additional Authenticated Data. Note that some
      algorithms may not use an Authentication Tag, in which case this
      value is the empty octet sequence.

   Content Encryption Key (CEK)
      A symmetric key for the AEAD algorithm used to encrypt the
      Plaintext to produce the Ciphertext and the Authentication Tag.

   JWE Encrypted Key
      Encrypted Content Encryption Key (CEK) value. Note that for some
      algorithms, the JWE Encrypted Key value is specified as being the
      empty octet sequence.

JWE Initialization Vector
   Initialization vector value used when encrypting the plaintext.
   Note that some algorithms may not use an Initialization Vector, in
   which case this value is the empty octet sequence.

JWE AAD
   Additional value to be integrity protected by the authenticated
   encryption operation. This can only be present when using the JWE
   JSON Serialization. (Note that this can also be achieved when
   using either serialization by including the AAD value as an
   integrity protected Header Parameter value, but at the cost of the
   value being double base64url encoded.)

JWE Ciphertext
   Ciphertext value resulting from authenticated encryption of the
   plaintext with additional authenticated data.

JWE Authentication Tag
   Authentication Tag value resulting from authenticated encryption
   of the plaintext with additional authenticated data.

JWE Protected Header
   JSON object that contains the Header Parameters that are integrity
   protected by the authenticated encryption operation. These
   parameters apply to all recipients of the JWE. For the JWE
   Compact Serialization, this comprises the entire JOSE Header. For
   the JWE JSON Serialization, this is one component of the JOSE
   Header.

JWE Shared Unprotected Header
   JSON object that contains the Header Parameters that apply to all
   recipients of the JWE that are not integrity protected. This can
   only be present when using the JWE JSON Serialization.

JWE Per-Recipient Unprotected Header
   JSON object that contains Header Parameters that apply to a single
   recipient of the JWE. These Header Parameter values are not
   integrity protected. This can only be present when using the JWE
   JSON Serialization.

JWE Compact Serialization
   A representation of the JWE as a compact, URL-safe string.

JWE JSON Serialization
   A representation of the JWE as a JSON object. The JWE JSON
   Serialization enables the same content to be encrypted to multiple
   parties. This representation is neither optimized for compactness
   nor URL-safe.

Key Management Mode
   A method of determining the Content Encryption Key (CEK) value to
   use. Each algorithm used for determining the CEK value uses a
   specific Key Management Mode. Key Management Modes employed by
   this specification are Key Encryption, Key Wrapping, Direct Key
   Agreement, Key Agreement with Key Wrapping, and Direct Encryption.

Key Encryption
   A Key Management Mode in which the Content Encryption Key (CEK)
   value is encrypted to the intended recipient using an asymmetric
   encryption algorithm.

Key Wrapping
   A Key Management Mode in which the Content Encryption Key (CEK)
   value is encrypted to the intended recipient using a symmetric key
   wrapping algorithm.

Direct Key Agreement
   A Key Management Mode in which a key agreement algorithm is used
   to agree upon the Content Encryption Key (CEK) value.

Key Agreement with Key Wrapping
   A Key Management Mode in which a key agreement algorithm is used
   to agree upon a symmetric key used to encrypt the Content
   Encryption Key (CEK) value to the intended recipient using a
   symmetric key wrapping algorithm.

Direct Encryption
   A Key Management Mode in which the Content Encryption Key (CEK)
   value used is the secret symmetric key value shared between the
   parties.

3. JSON Web Encryption (JWE) Overview

   JWE represents encrypted content using JSON data structures and
   base64url encoding. These JSON data structures MAY contain white
   space and/or line breaks. A JWE represents these logical values
   (each of which is defined in Section 2):

   o JOSE Header
   o JWE Encrypted Key
   o JWE Initialization Vector
   o JWE AAD
   o JWE Ciphertext
   o JWE Authentication Tag

   For a JWE object, the JOSE Header members are the union of the

members of these values (each of which is defined in Section 2):

o JWE Protected Header
o JWE Shared Unprotected Header
o JWE Per-Recipient Unprotected Header

JWE utilizes authenticated encryption to ensure the confidentiality
and integrity of the Plaintext and the integrity of the JWE Protected
Header and the JWE AAD.

This document defines two serializations for JWE objects: a compact,
URL-safe serialization called the JWE Compact Serialization and a
JSON serialization called the JWE JSON Serialization. In both
serializations, the JWE Protected Header, JWE Encrypted Key, JWE
Initialization Vector, JWE Ciphertext, and JWE Authentication Tag are
base64url encoded for transmission, since JSON lacks a way to
directly represent arbitrary octet sequences. When present, the JWE
AAD is also base64url encoded for transmission.

3.1. JWE Compact Serialization Overview

In the JWE Compact Serialization, no JWE Shared Unprotected Header or
JWE Per-Recipient Unprotected Header are used. In this case, the
JOSE Header and the JWE Protected Header are the same.

In the JWE Compact Serialization, a JWE object is represented as the
concatenation:

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||
BASE64URL(JWE Authentication Tag)
```

3.2. JWE JSON Serialization Overview

In the JWE JSON Serialization, one or more of the JWE Protected
Header, JWE Shared Unprotected Header, and JWE Per-Recipient
Unprotected Header MUST be present. In this case, the members of the
JOSE Header are the combination of the members of the JWE Protected
Header, JWE Shared Unprotected Header, and JWE Per-Recipient
Unprotected Header values that are present.

In the JWE JSON Serialization, a JWE object is represented as the
combination of these eight values,
```
BASE64URL(UTF8(JWE Protected Header)),
```

     JWE Shared Unprotected Header,
     JWE Per-Recipient Unprotected Header,
     BASE64URL(JWE Encrypted Key),
     BASE64URL(JWE Initialization Vector),
     BASE64URL(JWE Ciphertext),
     BASE64URL(JWE Authentication Tag), and
     BASE64URL(JWE AAD),
   with the six base64url encoded result strings and the two unprotected
   JSON object values being represented as members within a JSON object.
   The inclusion of some of these values is OPTIONAL. The JWE JSON
   Serialization can also encrypt the plaintext to multiple recipients.
   See Section 7.2 for more information about the JWE JSON
   Serialization.

3.3. Example JWE

   This example encrypts the plaintext "The true sign of intelligence is
   not knowledge but imagination." to the recipient.

   The following example JWE Protected Header declares that:
   o the Content Encryption Key is encrypted to the recipient using the
      RSAES OAEP [RFC3447] algorithm to produce the JWE Encrypted Key
      and
   o authenticated encryption is performed on the Plaintext using the
      AES GCM [AES, NIST.800-38D] algorithm with a 256 bit key to
      produce the Ciphertext and the Authentication Tag.

      {"alg":"RSA-OAEP","enc":"A256GCM"}

   Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected
   Header)) gives this value:

      eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ

   The remaining steps to finish creating this JWE are:
   o Generate a random Content Encryption Key (CEK).
   o Encrypt the CEK with the recipient's public key using the RSAES
      OAEP algorithm to produce the JWE Encrypted Key.
   o Base64url encode the JWE Encrypted Key.
   o Generate a random JWE Initialization Vector.
   o Base64url encode the JWE Initialization Vector.
   o Let the Additional Authenticated Data encryption parameter be
      ASCII(BASE64URL(UTF8(JWE Protected Header))).
   o Perform authenticated encryption on the Plaintext with the AES GCM
      algorithm using the CEK as the encryption key, the JWE
      Initialization Vector, and the Additional Authenticated Data
      value, requesting a 128 bit Authentication Tag output.

o Base64url encode the Ciphertext.
o Base64url encode the Authentication Tag.
o Assemble the final representation: The Compact Serialization of
   this result is the string BASE64URL(UTF8(JWE Protected Header)) ||
   '.' || BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE
   Initialization Vector) || '.' || BASE64URL(JWE Ciphertext) || '.'
   || BASE64URL(JWE Authentication Tag).

The final result in this example (with line breaks for display
purposes only) is:

eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg.
48V1_ALb6US04U3b.
5eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
SdiwkIr3ajwQzaBtQD_A.
XFBoMYUZodetZdvTiFvSkQ

See Appendix A.1 for the complete details of computing this JWE. See
other parts of Appendix A for additional examples.

4. JOSE Header

For a JWE object, the members of the JSON object(s) representing the
JOSE Header describe the encryption applied to the Plaintext and
optionally additional properties of the JWE. The Header Parameter
names within the JOSE Header MUST be unique, just as described in
Section 4 of [JWS]. The rules about handling Header Parameters that
are not understood by the implementation are also the same. The
classes of Header Parameter names are likewise the same.

4.1. Registered Header Parameter Names

The following Header Parameter names for use in JWE objects are
registered in the IANA JSON Web Signature and Encryption Header
Parameters registry defined in [JWS], with meanings as defined below.

As indicated by the common registry, JWSs and JWEs share a common
Header Parameter space; when a parameter is used by both
specifications, its usage must be compatible between the
specifications.

4.1.1. "alg" (Algorithm) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "alg" Header Parameter defined in Section 4.1.1 of [JWS], except
that the Header Parameter identifies the cryptographic algorithm used
to encrypt or determine the value of the Content Encryption Key
(CEK). The encrypted content is not usable if the "alg" value does
not represent a supported algorithm, or if the recipient does not
have a key that can be used with that algorithm.

A list of defined "alg" values for this use can be found in the IANA
JSON Web Signature and Encryption Algorithms registry defined in
[JWA]; the initial contents of this registry are the values defined
in Section 4.1 of the JSON Web Algorithms (JWA) [JWA] specification.

4.1.2. "enc" (Encryption Algorithm) Header Parameter

The "enc" (encryption algorithm) Header Parameter identifies the
content encryption algorithm used to perform authenticated encryption
on the Plaintext to produce the Ciphertext and the Authentication
Tag. This algorithm MUST be an AEAD algorithm with a specified key
length. The recipient MUST reject the JWE if the "enc" value does
not represent a supported algorithm. "enc" values should either be
registered in the IANA JSON Web Signature and Encryption Algorithms
registry defined in [JWA] or be a value that contains a Collision-
Resistant Name. The "enc" value is a case-sensitive string
containing a StringOrURI value. This Header Parameter MUST be
present and MUST be understood and processed by implementations.

A list of defined "enc" values for this use can be found in the IANA
JSON Web Signature and Encryption Algorithms registry defined in
[JWA]; the initial contents of this registry are the values defined
in Section 5.1 of the JSON Web Algorithms (JWA) [JWA] specification.

4.1.3. "zip" (Compression Algorithm) Header Parameter

The "zip" (compression algorithm) applied to the Plaintext before
encryption, if any. The "zip" value defined by this specification
is:

o   "DEF" - Compression with the DEFLATE [RFC1951] algorithm

Other values MAY be used. Compression algorithm values can be
registered in the IANA JSON Web Encryption Compression Algorithm
registry defined in [JWA]. The "zip" value is a case-sensitive
string. If no "zip" parameter is present, no compression is applied
to the Plaintext before encryption. This Header Parameter MUST be
integrity protected, and therefore MUST occur only within the JWE

Protected Header, when used. Use of this Header Parameter is
OPTIONAL. This Header Parameter MUST be understood and processed by
implementations.

4.1.4. "jku" (JWK Set URL) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "jku" Header Parameter defined in Section 4.1.2 of [JWS], except
that the JWK Set resource contains the public key to which the JWE
was encrypted; this can be used to determine the private key needed
to decrypt the JWE.

4.1.5. "jwk" (JSON Web Key) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "jwk" Header Parameter defined in Section 4.1.3 of [JWS], except
that the key is the public key to which the JWE was encrypted; this
can be used to determine the private key needed to decrypt the JWE.

4.1.6. "kid" (Key ID) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "kid" Header Parameter defined in Section 4.1.4 of [JWS], except
that the key hint references the public key to which the JWE was
encrypted; this can be used to determine the private key needed to
decrypt the JWE. This parameter allows originators to explicitly
signal a change of key to JWE recipients.

4.1.7. "x5u" (X.509 URL) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "x5u" Header Parameter defined in Section 4.1.5 of [JWS], except
that the X.509 public key certificate or certificate chain [RFC5280]
contains the public key to which the JWE was encrypted; this can be
used to determine the private key needed to decrypt the JWE.

4.1.8. "x5c" (X.509 Certificate Chain) Header Parameter

This parameter has the same meaning, syntax, and processing rules as
the "x5c" Header Parameter defined in Section 4.1.6 of [JWS], except
that the X.509 public key certificate or certificate chain [RFC5280]
contains the public key to which the JWE was encrypted; this can be
used to determine the private key needed to decrypt the JWE.

See Appendix B of [JWS] for an example "x5c" value.

4.1.9. "x5t" (X.509 Certificate SHA-1 Thumbprint) Header Parameter

   This parameter has the same meaning, syntax, and processing rules as
   the "x5t" Header Parameter defined in Section 4.1.7 of [JWS], except
   that the certificate referenced by the thumbprint contains the public
   key to which the JWE was encrypted; this can be used to determine the
   private key needed to decrypt the JWE. Note that certificate
   thumbprints are also sometimes known as certificate fingerprints.

4.1.10. "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) Header Parameter

   This parameter has the same meaning, syntax, and processing rules as
   the "x5t#S256" Header Parameter defined in Section 4.1.8 of [JWS],
   except that the certificate referenced by the thumbprint contains the
   public key to which the JWE was encrypted; this can be used to
   determine the private key needed to decrypt the JWE. Note that
   certificate thumbprints are also sometimes known as certificate
   fingerprints.

4.1.11. "typ" (Type) Header Parameter

   This parameter has the same meaning, syntax, and processing rules as
   the "typ" Header Parameter defined in Section 4.1.9 of [JWS], except
   that the type is that of this complete JWE object.

4.1.12. "cty" (Content Type) Header Parameter

   This parameter has the same meaning, syntax, and processing rules as
   the "cty" Header Parameter defined in Section 4.1.10 of [JWS], except
   that the type is that of the secured content (the plaintext).

4.1.13. "crit" (Critical) Header Parameter

   This parameter has the same meaning, syntax, and processing rules as
   the "crit" Header Parameter defined in Section 4.1.11 of [JWS],
   except that Header Parameters for a JWE object are being referred to,
   rather than Header Parameters for a JWS object.

4.2. Public Header Parameter Names

   Additional Header Parameter names can be defined by those using JWEs.
   However, in order to prevent collisions, any new Header Parameter
   name should either be registered in the IANA JSON Web Signature and
   Encryption Header Parameters registry defined in [JWS] 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 Header Parameter name.

New Header Parameters should be introduced sparingly, as they can
result in non-interoperable JWEs.

4.3. Private Header Parameter Names

A producer and consumer of a JWE may agree to use Header Parameter
names that are Private Names: names that are not Registered Header
Parameter names Section 4.1 or Public Header Parameter names
Section 4.2. Unlike Public Header Parameter names, Private Header
Parameter names are subject to collision and should be used with
caution.

5. Producing and Consuming JWEs

5.1. Message Encryption

The message encryption process is as follows. The order of the steps
is not significant in cases where there are no dependencies between
the inputs and outputs of the steps.

1. Determine the Key Management Mode employed by the algorithm used
    to determine the Content Encryption Key (CEK) value. (This is
    the algorithm recorded in the "alg" (algorithm) Header Parameter
    of the resulting JWE.)

2. When Key Wrapping, Key Encryption, or Key Agreement with Key
    Wrapping are employed, generate a random Content Encryption Key
    (CEK) value. See RFC 4086 [RFC4086] for considerations on
    generating random values. The CEK MUST have a length equal to
    that required for the content encryption algorithm.

3. When Direct Key Agreement or Key Agreement with Key Wrapping are
    employed, use the key agreement algorithm to compute the value
    of the agreed upon key. When Direct Key Agreement is employed,
    let the Content Encryption Key (CEK) be the agreed upon key.
    When Key Agreement with Key Wrapping is employed, the agreed
    upon key will be used to wrap the CEK.

4. When Key Wrapping, Key Encryption, or Key Agreement with Key
    Wrapping are employed, encrypt the CEK to the recipient and let
    the result be the JWE Encrypted Key.

5. When Direct Key Agreement or Direct Encryption are employed, let
    the JWE Encrypted Key be the empty octet sequence.

6.   When Direct Encryption is employed, let the Content Encryption
      Key (CEK) be the shared symmetric key.

7.   Compute the encoded key value BASE64URL(JWE Encrypted Key).

8.   If the JWE JSON Serialization is being used, repeat this process
      (steps 1-7) for each recipient.

9.   Generate a random JWE Initialization Vector of the correct size
      for the content encryption algorithm (if required for the
      algorithm); otherwise, let the JWE Initialization Vector be the
      empty octet sequence.

10.   Compute the encoded initialization vector value BASE64URL(JWE
      Initialization Vector).

11.   If a "zip" parameter was included, compress the Plaintext using
      the specified compression algorithm and let M be the octet
      sequence representing the compressed Plaintext; otherwise, let M
      be the octet sequence representing the Plaintext.

12.   Create the JSON object(s) containing the desired set of Header
      Parameters, which together comprise the JOSE Header: the JWE
      Protected Header, and if the JWE JSON Serialization is being
      used, the JWE Shared Unprotected Header and the JWE Per-
      Recipient Unprotected Header.

13.   Compute the Encoded Protected Header value BASE64URL(UTF8(JWE
      Protected Header)). If the JWE Protected Header is not present
      (which can only happen when using the JWE JSON Serialization and
      no "protected" member is present), let this value be the empty
      string.

14.   Let the Additional Authenticated Data encryption parameter be
      ASCII(Encoded Protected Header). However if a JWE AAD value is
      present (which can only be the case when using the JWE JSON
      Serialization), instead let the Additional Authenticated Data
      encryption parameter be ASCII(Encoded Protected Header || '.' ||
      BASE64URL(JWE AAD)).

15.   Encrypt M using the CEK, the JWE Initialization Vector, and the
      Additional Authenticated Data value using the specified content
      encryption algorithm to create the JWE Ciphertext value and the
      JWE Authentication Tag (which is the Authentication Tag output
      from the encryption operation).

16.   Compute the encoded ciphertext value BASE64URL(JWE Ciphertext).

17. Compute the encoded authentication tag value BASE64URL(JWE
    Authentication Tag).

18. The five encoded values are used in both the JWE Compact
    Serialization and the JWE JSON Serialization representations.

19. If a JWE AAD value is present, compute the encoded AAD value
    BASE64URL(JWE AAD).

20. Create the desired serialized output. The Compact Serialization
    of this result is the string BASE64URL(UTF8(JWE Protected
    Header)) || '.' || BASE64URL(JWE Encrypted Key) || '.' ||
    BASE64URL(JWE Initialization Vector) || '.' || BASE64URL(JWE
    Ciphertext) || '.' || BASE64URL(JWE Authentication Tag). The
    JWE JSON Serialization is described in Section 7.2.

5.2. Message Decryption

The message decryption process is the reverse of the encryption
process. 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 these steps fails, the encrypted content cannot be
validated.

When there are multiple recipients, it is an application decision
which of the recipients' encrypted content must successfully validate
for the JWE to be accepted. In some cases, encrypted content for all
recipients must successfully validate or the JWE will be rejected.
In other cases, only the encrypted content for a single recipient
needs to be successfully validated. However, in all cases, the
encrypted content for at least one recipient MUST successfully
validate or the JWE MUST be rejected.

1. Parse the JWE representation to extract the serialized values
    for the components of the JWE. When using the JWE Compact
    Serialization, these components are the base64url encoded
    representations of the JWE Protected Header, the JWE Encrypted
    Key, the JWE Initialization Vector, the JWE Ciphertext, and the
    JWE Authentication Tag, and when using the JWE JSON
    Serialization, these components also include the base64url
    encoded representation of the JWE AAD and the unencoded JWE
    Shared Unprotected Header and JWE Per-Recipient Unprotected
    Header values. When using the JWE Compact Serialization, the
    JWE Protected Header, the JWE Encrypted Key, the JWE
    Initialization Vector, the JWE Ciphertext, and the JWE
    Authentication Tag are represented as base64url encoded values
    in that order, separated by four period ('.') characters. The
    JWE JSON Serialization is described in Section 7.2.

2.   The encoded representations of the JWE Protected Header, the JWE
     Encrypted Key, the JWE Initialization Vector, the JWE
     Ciphertext, the JWE Authentication Tag, and the JWE AAD MUST be
     successfully base64url decoded following the restriction that no
     padding characters have been used.

3.   The octet sequence resulting from decoding the encoded JWE
     Protected Header MUST be a UTF-8 encoded representation of a
     completely valid JSON object conforming to RFC 7159 [RFC7159],
     which is the JWE Protected Header.

4.   If using the JWE Compact Serialization, let the JOSE Header be
     the JWE Protected Header. Otherwise, when using the JWE JSON
     Serialization, let the JOSE Header be the union of the members
     of the JWE Protected Header, the JWE Shared Unprotected Header
     and the corresponding JWE Per-Recipient Unprotected Header, all
     of which must be completely valid JSON objects.

5.   The resulting JOSE Header MUST NOT contain duplicate Header
     Parameter names. When using the JWE JSON Serialization, this
     restriction includes that the same Header Parameter name also
     MUST NOT occur in distinct JSON object values that together
     comprise the JOSE Header.

6.   Verify that the implementation understands and can process all
     fields that it is required to support, whether required by this
     specification, by the algorithms being used, or by the "crit"
     Header Parameter value, and that the values of those parameters
     are also understood and supported.

7.   Determine the Key Management Mode employed by the algorithm
     specified by the "alg" (algorithm) Header Parameter.

8.   Verify that the JWE uses a key known to the recipient.

9.   When Direct Key Agreement or Key Agreement with Key Wrapping are
     employed, use the key agreement algorithm to compute the value
     of the agreed upon key. When Direct Key Agreement is employed,
     let the Content Encryption Key (CEK) be the agreed upon key.
     When Key Agreement with Key Wrapping is employed, the agreed
     upon key will be used to decrypt the JWE Encrypted Key.

10. When Key Wrapping, Key Encryption, or Key Agreement with Key
     Wrapping are employed, decrypt the JWE Encrypted Key to produce
     the Content Encryption Key (CEK). The CEK MUST have a length
     equal to that required for the content encryption algorithm.
     Note that when there are multiple recipients, each recipient
     will only be able decrypt any JWE Encrypted Key values that were

encrypted to a key in that recipient's possession. It is
therefore normal to only be able to decrypt one of the per-
recipient JWE Encrypted Key values to obtain the CEK value.
Also, see [Section 11.5](#) for security considerations on mitigating
timing attacks.

11. When Direct Key Agreement or Direct Encryption are employed,
     verify that the JWE Encrypted Key value is empty octet sequence.

12. When Direct Encryption is employed, let the Content Encryption
     Key (CEK) be the shared symmetric key.

13. Record whether the CEK could be successfully determined for this
     recipient or not.

14. If the JWE JSON Serialization is being used, repeat this process
     (steps 4-13) for each recipient contained in the representation.

15. Compute the Encoded Protected Header value BASE64URL(UTF8(JWE
     Protected Header)). If the JWE Protected Header is not present
     (which can only happen when using the JWE JSON Serialization and
     no "protected" member is present), let this value be the empty
     string.

16. Let the Additional Authenticated Data encryption parameter be
     ASCII(Encoded Protected Header). However if a JWE AAD value is
     present (which can only be the case when using the JWE JSON
     Serialization), instead let the Additional Authenticated Data
     encryption parameter be ASCII(Encoded Protected Header || '.' ||
     BASE64URL(JWE AAD)).

17. Decrypt the JWE Ciphertext using the CEK, the JWE Initialization
     Vector, the Additional Authenticated Data value, and the JWE
     Authentication Tag (which is the Authentication Tag input to the
     calculation) using the specified content encryption algorithm,
     returning the decrypted plaintext and validating the JWE
     Authentication Tag in the manner specified for the algorithm,
     rejecting the input without emitting any decrypted output if the
     JWE Authentication Tag is incorrect.

18. If a "zip" parameter was included, uncompress the decrypted
     plaintext using the specified compression algorithm.

19. If there was no recipient for which all of the decryption steps
     succeeded, then the JWE MUST be rejected. Otherwise, output the
     Plaintext. In the JWE JSON Serialization case, also return a
     result to the application indicating for which of the recipients
     the decryption succeeded and failed.

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

5.3. String Comparison Rules

The string comparison rules for this specification are the same as
those defined in Section 5.3 of [JWS].

6. Key Identification

The key identification methods for this specification are the same as
those defined in Section 6 of [JWS], except that the key being
identified is the public key to which the JWE was encrypted.

7. Serializations

JWE objects use one of two serializations, the JWE Compact
Serialization or the JWE JSON Serialization. Applications using this
specification need to specify what serialization and serialization
features are used for that application. For instance, applications
might specify that only the JWE JSON Serialization is used, that only
JWE JSON Serialization support for a single recipient is used, or
that support for multiple recipients is used. JWE implementations
only need to implement the features needed for the applications they
are designed to support.

7.1. JWE Compact Serialization

The JWE Compact Serialization represents encrypted content as a
compact, URL-safe string. This string is:

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||
BASE64URL(JWE Authentication Tag)
```

Only one recipient is supported by the JWE Compact Serialization and
it provides no syntax to represent JWE Shared Unprotected Header, JWE
Per-Recipient Unprotected Header, or JWE AAD values.

7.2. **JWE JSON Serialization**

   The JWE JSON Serialization represents encrypted content as a JSON
   object. Content using the JWE JSON Serialization can be encrypted to
   more than one recipient. This representation is neither optimized
   for compactness nor URL-safe.

   The following members are defined for use in top-level JSON objects
   used for the JWE JSON Serialization:

   protected
      The "protected" member MUST be present and contain the value
      BASE64URL(UTF8(JWE Protected Header)) when the JWE Protected
      Header value is non-empty; otherwise, it MUST be absent. These
      Header Parameter values are integrity protected.

   unprotected
      The "unprotected" member MUST be present and contain the value JWE
      Shared Unprotected Header when the JWE Shared Unprotected Header
      value is non-empty; otherwise, it MUST be absent. This value is
      represented as an unencoded JSON object, rather than as a string.
      These Header Parameter values are not integrity protected.

   iv
      The "iv" member MUST be present and contain the value
      BASE64URL(JWE Initialization Vector) when the JWE Initialization
      Vector value is non-empty; otherwise, it MUST be absent.

   aad
      The "aad" member MUST be present and contain the value
      BASE64URL(JWE AAD)) when the JWE AAD value is non-empty;
      otherwise, it MUST be absent. A JWE AAD value can be included to
      supply a base64url encoded value to be integrity protected but not
      encrypted.

   ciphertext
      The "ciphertext" member MUST be present and contain the value
      BASE64URL(JWE Ciphertext).

   tag
      The "tag" member MUST be present and contain the value
      BASE64URL(JWE Authentication Tag) when the JWE Authentication Tag
      value is non-empty; otherwise, it MUST be absent.

   recipients
      The "recipients" member value MUST be an array of JSON objects.
      Each object contains information specific to a single recipient.
      This member MUST be present, even if the array elements contain

only the empty JSON object "{}" (which can happen when all Header
Parameter values are shared between all recipients and when no
encrypted key is used, such as when doing Direct Encryption).

The following members are defined for use in the JSON objects that
are elements of the "recipients" array:

header
    The "header" member MUST be present and contain the value JWE Per-
    Recipient Unprotected Header when the JWE Per-Recipient
    Unprotected Header value is non-empty; otherwise, it MUST be
    absent. This value is represented as an unencoded JSON object,
    rather than as a string. These Header Parameter values are not
    integrity protected.

encrypted_key
    The "encrypted_key" member MUST be present and contain the value
    BASE64URL(JWE Encrypted Key) when the JWE Encrypted Key value is
    non-empty; otherwise, it MUST be absent.

At least one of the "header", "protected", and "unprotected" members
MUST be present so that "alg" and "enc" Header Parameter values are
conveyed for each recipient computation.

Additional members can be present in both the JSON objects defined
above; if not understood by implementations encountering them, they
MUST be ignored.

Some Header Parameters, including the "alg" parameter, can be shared
among all recipient computations. Header Parameters in the JWE
Protected Header and JWE Shared Unprotected Header values are shared
among all recipients.

The Header Parameter values used when creating or validating per-
recipient Ciphertext and Authentication Tag values are the union of
the three sets of Header Parameter values that may be present: (1)
the JWE Protected Header represented in the "protected" member, (2)
the JWE Shared Unprotected Header represented in the "unprotected"
member, and (3) the JWE Per-Recipient Unprotected Header represented
in the "header" member of the recipient's array element. The union
of these sets of Header Parameters comprises the JOSE Header. The
Header Parameter names in the three locations MUST be disjoint.

Each JWE Encrypted Key value is computed using the parameters of the
corresponding JOSE Header value in the same manner as for the JWE
Compact Serialization. This has the desirable property that each JWE
Encrypted Key value in the "recipients" array is identical to the
value that would have been computed for the same parameter in the JWE

Compact Serialization. Likewise, the JWE Ciphertext and JWE
Authentication Tag values match those produced for the JWE Compact
Serialization, provided that the JWE Protected Header value (which
represents the integrity-protected Header Parameter values) matches
that used in the JWE Compact Serialization.

All recipients use the same JWE Protected Header, JWE Initialization
Vector, JWE Ciphertext, and JWE Authentication Tag values, when
present, resulting in potentially significant space savings if the
message is large. Therefore, all Header Parameters that specify the
treatment of the Plaintext value MUST be the same for all recipients.
This primarily means that the "enc" (encryption algorithm) Header
Parameter value in the JOSE Header for each recipient and any
parameters of that algorithm MUST be the same.

In summary, the syntax of a JWE using the JWE JSON Serialization is
as follows:

```
{"protected":"<integrity-protected shared header contents>",
 "unprotected":<non-integrity-protected shared header contents>,
 "recipients":[
  {"header":<per-recipient unprotected header 1 contents>,
   "encrypted_key":"<encrypted key 1 contents>"},
  ...
  {"header":<per-recipient unprotected header N contents>,
   "encrypted_key":"<encrypted key N contents>"}],
 "aad":"<additional authenticated data contents>",
 "iv":"<initialization vector contents>",
 "ciphertext":"<ciphertext contents>",
 "tag":"<authentication tag contents>"
}
```

See Appendix A.4 for an example of computing a JWE using the JWE JSON
Serialization.

8. TLS Requirements

The TLS requirements for this specification are the same as those
defined in Section 8 of [JWS].

9. Distinguishing between JWS and JWE Objects

There are several ways of distinguishing whether an object is a JWS
or JWE object. All these methods will yield the same result for all
legal input values; they may yield different results for malformed
inputs.

o If the object is using the JWS Compact Serialization or the JWE
   Compact Serialization, the number of base64url encoded segments
   separated by period ('.') characters differs for JWSs and JWEs.
   JWSs have three segments separated by two period ('.') characters.
   JWEs have five segments separated by four period ('.') characters.

o If the object is using the JWS JSON Serialization or the JWE JSON
   Serialization, the members used will be different. JWSs have a
   "signatures" member and JWEs do not. JWEs have a "recipients"
   member and JWSs do not.

o The JOSE Header for a JWS object can be distinguished from the
   JOSE Header for a JWE object by examining the "alg" (algorithm)
   Header Parameter value. If the value represents a digital
   signature or MAC algorithm, or is the value "none", it is for a
   JWS; if it represents a Key Encryption, Key Wrapping, Direct Key
   Agreement, Key Agreement with Key Wrapping, or Direct Encryption
   algorithm, it is for a JWE. (Extracting the "alg" value to
   examine is straightforward when using the JWS Compact
   Serialization or the JWE Compact Serialization and may be more
   difficult when using the JWS JSON Serialization or the JWE JSON
   Serialization.)

o The JOSE Header for a JWS object can also be distinguished from
   the JOSE Header for a JWE object by determining whether an "enc"
   (encryption algorithm) member exists. If the "enc" member exists,
   it is a JWE; otherwise, it is a JWS.

10. IANA Considerations

10.1. JSON Web Signature and Encryption Header Parameters Registration

   This specification registers the Header Parameter names defined in
   Section 4.1 in the IANA JSON Web Signature and Encryption Header
   Parameters registry defined in [JWS].

10.1.1. Registry Contents

o Header Parameter Name: "alg"
o Header Parameter Description: Algorithm
o Header Parameter Usage Location(s): JWE
o Change Controller: IESG
o Specification Document(s): Section 4.1.1 of [[this document]]

o Header Parameter Name: "enc"

   o Header Parameter Description: Encryption Algorithm
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.2 of [[this document]]

   o Header Parameter Name: "zip"
   o Header Parameter Description: Compression Algorithm
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.3 of [[this document]]

   o Header Parameter Name: "jku"
   o Header Parameter Description: JWK Set URL
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.4 of [[this document]]

   o Header Parameter Name: "jwk"
   o Header Parameter Description: JSON Web Key
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification document(s): Section 4.1.5 of [[this document]]

   o Header Parameter Name: "kid"
   o Header Parameter Description: Key ID
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.6 of [[this document]]

   o Header Parameter Name: "x5u"
   o Header Parameter Description: X.509 URL
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.7 of [[this document]]

   o Header Parameter Name: "x5c"
   o Header Parameter Description: X.509 Certificate Chain
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.8 of [[this document]]

   o Header Parameter Name: "x5t"
   o Header Parameter Description: X.509 Certificate SHA-1 Thumbprint
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.9 of [[this document]]

   o Header Parameter Name: "x5t#S256"
   o Header Parameter Description: X.509 Certificate SHA-256 Thumbprint
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.10 of [[this document]]

   o Header Parameter Name: "typ"
   o Header Parameter Description: Type
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.11 of [[this document]]

   o Header Parameter Name: "cty"
   o Header Parameter Description: Content Type
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.12 of [[this document]]

   o Header Parameter Name: "crit"
   o Header Parameter Description: Critical
   o Header Parameter Usage Location(s): JWE
   o Change Controller: IESG
   o Specification Document(s): Section 4.1.13 of [[this document]]

11. Security Considerations

   All of the security issues that are pertinent to any cryptographic
   application must be addressed by 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 this specification. Likewise, all the security considerations in
   XML Encryption 1.1 [W3C.REC-xmlenc-core1-20130411] also apply, other
   than those that are XML specific.

11.1. Key Entropy and Random Values

   See Section 10.1 of [JWS] for security considerations on key entropy
   and random values. In addition to the uses of random values listed
   there, note that random values are also used for content encryption
   keys (CEKs) and initialization vectors (IVs) when performing
   encryption.

11.2. **Key Protection**

   See Section 10.2 of [JWS] for security considerations on key
   protection. In addition to the keys listed there that must be
   protected, implementations performing encryption must protect the key
   encryption key and the content encryption key. Compromise of the key
   encryption key may result in the disclosure of all contents protected
   with that key. Similarly, compromise of the content encryption key
   may result in disclosure of the associated encrypted content.

11.3. **Using Matching Algorithm Strengths**

   Algorithms of matching strengths should be used together whenever
   possible. For instance, when AES Key Wrap is used with a given key
   size, using the same key size is recommended when AES GCM is also
   used. If the key encryption and content encryption algorithms are
   different, the effective security is determined by the weaker of the
   two algorithms.

   Also, see RFC 3766 [RFC3766] for information on determining strengths
   for public keys used for exchanging symmetric keys.

11.4. **Adaptive Chosen-Ciphertext Attacks**

   When decrypting, particular care must be taken not to allow the JWE
   recipient to be used as an oracle for decrypting messages. RFC 3218
   [RFC3218] should be consulted for specific countermeasures to attacks
   on RSAES-PKCS1-V1_5. An attacker might modify the contents of the
   "alg" parameter from "RSA-OAEP" to "RSA1_5" in order to generate a
   formatting error that can be detected and used to recover the CEK
   even if RSAES OAEP was used to encrypt the CEK. It is therefore
   particularly important to report all formatting errors to the CEK,
   Additional Authenticated Data, or ciphertext as a single error when
   the encrypted content is rejected.

   Additionally, this type of attack can be prevented by restricting the
   use of a key to a limited set of algorithms -- usually one. This
   means, for instance, that if the key is marked as being for
   "RSA-OAEP" only, any attempt to decrypt a message using the "RSA1_5"
   algorithm with that key should fail immediately due to invalid use of
   the key.

11.5. **Timing Attacks**

   To mitigate the attacks described in RFC 3218 [RFC3218], the
   recipient MUST NOT distinguish between format, padding, and length
   errors of encrypted keys. It is strongly recommended, in the event
   of receiving an improperly formatted key, that the receiver

substitute a randomly generated CEK and proceed to the next step, to
mitigate timing attacks.

12. References

12.1. Normative References

[JWA]   Jones, M., "JSON Web Algorithms (JWA)",
            draft-ietf-jose-json-web-algorithms (work in progress),
            September 2014.

[JWK]   Jones, M., "JSON Web Key (JWK)",
            draft-ietf-jose-json-web-key (work in progress),
            September 2014.

[JWS]   Jones, M., Bradley, J., and N. Sakimura, "JSON Web
            Signature (JWS)", draft-ietf-jose-json-web-signature (work
            in progress), September 2014.

[RFC1951]   Deutsch, P., "DEFLATE Compressed Data Format Specification
            version 1.3", RFC 1951, May 1996.

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

[RFC3629]   Yergeau, F., "UTF-8, a transformation format of ISO
            10646", STD 63, RFC 3629, November 2003.

[RFC5280]   Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
            Housley, R., and W. Polk, "Internet X.509 Public Key
            Infrastructure Certificate and Certificate Revocation List
            (CRL) Profile", RFC 5280, May 2008.

[RFC7159]   Bray, T., "The JavaScript Object Notation (JSON) Data
            Interchange Format", RFC 7159, March 2014.

[USASCII]   American National Standards Institute, "Coded Character
            Set -- 7-bit American Standard Code for Information
            Interchange", ANSI X3.4, 1986.

12.2. Informative References

[AES]   National Institute of Standards and Technology (NIST),
            "Advanced Encryption Standard (AES)", FIPS PUB 197,
            November 2001.

[I-D.mcgrew-aead-aes-cbc-hmac-sha2]

              McGrew, D., Foley, J., and K. Paterson, "Authenticated
              Encryption with AES-CBC and HMAC-SHA",
              draft-mcgrew-aead-aes-cbc-hmac-sha2-05 (work in progress),
              July 2014.

   [I-D.rescorla-jsms]
              Rescorla, E. and J. Hildebrand, "JavaScript Message
              Security Format", draft-rescorla-jsms-00 (work in
              progress), March 2011.

   [JSE]   Bradley, J. and N. Sakimura (editor), "JSON Simple
              Encryption", September 2010.

   [NIST.800-38D]
              National Institute of Standards and Technology (NIST),
              "Recommendation for Block Cipher Modes of Operation:
              Galois/Counter Mode (GCM) and GMAC", NIST PUB 800-38D,
              December 2001.

   [RFC3218] Rescorla, E., "Preventing the Million Message Attack on
              Cryptographic Message Syntax", RFC 3218, January 2002.

   [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography
              Standards (PKCS) #1: RSA Cryptography Specifications
              Version 2.1", RFC 3447, February 2003.

   [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For
              Public Keys Used For Exchanging Symmetric Keys", BCP 86,
              RFC 3766, April 2004.

   [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
              Requirements for Security", BCP 106, RFC 4086, June 2005.

   [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
              RFC 5652, September 2009.

   [W3C.REC-xmlenc-core1-20130411]
              Eastlake, D., Reagle, J., Hirsch, F., and T. Roessler,
              "XML Encryption Syntax and Processing Version 1.1", World
              Wide Web Consortium Recommendation REC-xmlenc-core1-
              20130411, April 2013,
              <http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/>.

Appendix A. JWE Examples

   This section provides examples of JWE computations.

A.1. Example JWE using RSAES OAEP and AES GCM

   This example encrypts the plaintext "The true sign of intelligence is
   not knowledge but imagination." to the recipient using RSAES OAEP for
   key encryption and AES GCM for content encryption. The
   representation of this plaintext (using JSON array notation) is:

   [84, 104, 101, 32, 116, 114, 117, 101, 32, 115, 105, 103, 110, 32,
   111, 102, 32, 105, 110, 116, 101, 108, 108, 105, 103, 101, 110, 99,
   101, 32, 105, 115, 32, 110, 111, 116, 32, 107, 110, 111, 119, 108,
   101, 100, 103, 101, 32, 98, 117, 116, 32, 105, 109, 97, 103, 105,
   110, 97, 116, 105, 111, 110, 46]

A.1.1. JOSE Header

   The following example JWE Protected Header declares that:

   o the Content Encryption Key is encrypted to the recipient using the
      RSAES OAEP algorithm to produce the JWE Encrypted Key and

   o authenticated encryption is performed on the Plaintext using the
      AES GCM algorithm with a 256 bit key to produce the Ciphertext and
      the Authentication Tag.

      {"alg":"RSA-OAEP","enc":"A256GCM"}

   Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected
   Header)) gives this value:

      eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ

A.1.2. Content Encryption Key (CEK)

   Generate a 256 bit random Content Encryption Key (CEK). In this
   example, the value (using JSON array notation) is:

   [177, 161, 244, 128, 84, 143, 225, 115, 63, 180, 3, 255, 107, 154,
   212, 246, 138, 7, 110, 91, 112, 46, 34, 105, 47, 130, 203, 46, 122,
   234, 64, 252]

A.1.3. Key Encryption

   Encrypt the CEK with the recipient's public key using the RSAES OAEP
   algorithm to produce the JWE Encrypted Key. This example uses the RSA
   key represented in JSON Web Key [JWK] format below (with line breaks
   within values for display purposes only):

    {"kty":"RSA",
     "n":"oahUIoWw0K0usKNuOR6H4wkf4oBUXHTxRvgb48E-BVvxkeDNjbC4he8rUW
           cJoZmds2h7M70imEVhRU5djINXtqllXI4DFqcI1DgjT9LewND8MW2Krf3S
           psk_ZkoFnilakGygTwpZ3uesH-PFABNIUYpOiN15dsQRkgr0vEhxN92i2a
           sbOenSZeyaxziK72UwxrrKoExv6kc5twXTq4h-QChLOln0_mtUZwfsRaMS
           tPs6mS6XrgxnxbWhojf663tuEQueGC-FCMfra36C9knDFGzKsNa7LZK2dj
           YgyD3JR_MB_4NUJW_TqOQtwHYbxevoJArm-L5StowjzGy-_bq6Gw",
     "e":"AQAB",
     "d":"kLdtIj6GbDks_ApCSTYQtelcNttlKiOyPzMrXHeI-yk1F7-kpDxY4-WY5N
           WV5KntaEeXS1j82E375xxhWMHXyvjYecPT9fpwR_M9gV8n9Hrh2anTpTD9
           3Dt62ypW3yDsJzBnTnrYu1iwWRgBKrEYY46qAZIrA2xAwnm2X7uGR1hghk
           qDp0Vqj3kbSCz1XyfCs6_LehBwtxHIyh8Ripy40p24moOAbgxVw3rxT_vl
           t3UVe4WO3JkJOzlpUf-KTVI2Ptgm-dARxTEtE-id-4OJr0h-K-VFs3VSnd
           VTIznSxfyrj8ILL6MG_Uv8YAu7VILSB3lOW085-4qE3DzgrTjgyQ"
    }

The resulting JWE Encrypted Key value is:

[56, 163, 154, 192, 58, 53, 222, 4, 105, 218, 136, 218, 29, 94, 203,
22, 150, 92, 129, 94, 211, 232, 53, 89, 41, 60, 138, 56, 196, 216,
82, 98, 168, 76, 37, 73, 70, 7, 36, 8, 191, 100, 136, 196, 244, 220,
145, 158, 138, 155, 4, 117, 141, 230, 199, 247, 173, 45, 182, 214,
74, 177, 107, 211, 153, 11, 205, 196, 171, 226, 162, 128, 171, 182,
13, 237, 239, 99, 193, 4, 91, 219, 121, 223, 107, 167, 61, 119, 228,
173, 156, 137, 134, 200, 80, 219, 74, 253, 56, 185, 91, 177, 34, 158,
89, 154, 205, 96, 55, 18, 138, 43, 96, 218, 215, 128, 124, 75, 138,
243, 85, 25, 109, 117, 140, 26, 155, 249, 67, 167, 149, 231, 100, 6,
41, 65, 214, 251, 232, 87, 72, 40, 182, 149, 154, 168, 31, 193, 126,
215, 89, 28, 111, 219, 125, 182, 139, 235, 195, 197, 23, 234, 55, 58,
63, 180, 68, 202, 206, 149, 75, 205, 248, 176, 67, 39, 178, 60, 98,
193, 32, 238, 122, 96, 158, 222, 57, 183, 111, 210, 55, 188, 215,
206, 180, 166, 150, 166, 106, 250, 55, 229, 72, 40, 69, 214, 216,
104, 23, 40, 135, 212, 28, 127, 41, 80, 175, 174, 168, 115, 171, 197,
89, 116, 92, 103, 246, 83, 216, 182, 176, 84, 37, 147, 35, 45, 219,
172, 99, 226, 233, 73, 37, 124, 42, 72, 49, 242, 35, 127, 184, 134,
117, 114, 135, 206]

Encoding this JWE Encrypted Key as BASE64URL(JWE Encrypted Key) gives
this value (with line breaks for display purposes only):

    OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
    ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
    Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
    mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
    1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
    6UklfCpIMfIjf7iGdXKHzg

[A.1.4](#). **Initialization Vector**

   Generate a random 96 bit JWE Initialization Vector. In this example,
   the value is:

   [227, 197, 117, 252, 2, 219, 233, 68, 180, 225, 77, 219]

   Encoding this JWE Initialization Vector as BASE64URL(JWE
   Initialization Vector) gives this value:

     48V1_ALb6US04U3b

[A.1.5](#). **Additional Authenticated Data**

   Let the Additional Authenticated Data encryption parameter be
   ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:

   [101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 83, 85, 48, 69,
   116, 84, 48, 70, 70, 85, 67, 73, 115, 73, 109, 86, 117, 89, 121, 73,
   54, 73, 107, 69, 121, 78, 84, 90, 72, 81, 48, 48, 105, 102, 81]

[A.1.6](#). **Content Encryption**

   Perform authenticated encryption on the Plaintext with the AES GCM
   algorithm using the CEK as the encryption key, the JWE Initialization
   Vector, and the Additional Authenticated Data value above, requesting
   a 128 bit Authentication Tag output. The resulting Ciphertext is:

   [229, 236, 166, 241, 53, 191, 115, 196, 174, 43, 73, 109, 39, 122,
   233, 96, 140, 206, 120, 52, 51, 237, 48, 11, 190, 219, 186, 80, 111,
   104, 50, 142, 47, 167, 59, 61, 181, 127, 196, 21, 40, 82, 242, 32,
   123, 143, 168, 226, 73, 216, 176, 144, 138, 247, 106, 60, 16, 205,
   160, 109, 64, 63, 192]

   The resulting Authentication Tag value is:

   [92, 80, 104, 49, 133, 25, 161, 215, 173, 101, 219, 211, 136, 91,
   210, 145]

   Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this
   value (with line breaks for display purposes only):

     5eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
     SdiwkIr3ajwQzaBtQD_A

   Encoding this JWE Authentication Tag as BASE64URL(JWE Authentication
   Tag) gives this value:

XFBoMYUZodetZdvTiFvSkQ

A.1.7. Complete Representation

Assemble the final representation: The Compact Serialization of this
result is the string BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE Initialization
Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE
Authentication Tag).

The final result in this example (with line breaks for display
purposes only) is:

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.
OKOawDo13gRp2ojaHV7LFpZcgV7T6DVZKTyKOMTYUmKoTCVJRgckCL9kiMT03JGe
ipsEdY3mx_etLbbWSrFr05kLzcSr4qKAq7YN7e9jwQRb23nfa6c9d-StnImGyFDb
Sv04uVuxIp5Zms1gNxKKK2Da14B8S4rzVRltdYwam_lDp5XnZAYpQdb76FdIKLaV
mqgfwX7XWRxv2322i-vDxRfqNzo_tETKzpVLzfiwQyeyPGLBIO56YJ7eObdv0je8
1860ppamavo35UgoRdbYaBcoh9QcfylQr66oc6vFWXRcZ_ZT2LawVCWTIy3brGPi
6UklfCpIMfIjf7iGdXKHzg.
48V1_ALb6US04U3b.
5eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs9tX_EFShS8iB7j6ji
SdiwkIr3ajwQzaBtQD_A.
XFBoMYUZodetZdvTiFvSkQ
```

A.1.8. Validation

This example illustrates the process of creating a JWE with RSAES
OAEP for key encryption and AES GCM for content encryption. These
results can be used to validate JWE decryption implementations for
these algorithms. Note that since the RSAES OAEP computation
includes random values, the encryption results above will not be
completely reproducible. However, since the AES GCM computation is
deterministic, the JWE Encrypted Ciphertext values will be the same
for all encryptions performed using these inputs.

A.2. Example JWE using RSAES-PKCS1-V1_5 and AES_128_CBC_HMAC_SHA_256

This example encrypts the plaintext "Live long and prosper." to the
recipient using RSAES-PKCS1-V1_5 for key encryption and
AES_128_CBC_HMAC_SHA_256 for content encryption. The representation
of this plaintext (using JSON array notation) is:

[76, 105, 118, 101, 32, 108, 111, 110, 103, 32, 97, 110, 100, 32,
112, 114, 111, 115, 112, 101, 114, 46]

A.2.1. **JOSE Header**

The following example JWE Protected Header declares that:

o the Content Encryption Key is encrypted to the recipient using the
   RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key and

o authenticated encryption is performed on the Plaintext using the
   AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext and
   the Authentication Tag.

   {"alg":"RSA1_5","enc":"A128CBC-HS256"}

Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected
Header)) gives this value:

   eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0

A.2.2. **Content Encryption Key (CEK)**

Generate a 256 bit random Content Encryption Key (CEK). In this
example, the key value is:

[4, 211, 31, 197, 84, 157, 252, 254, 11, 100, 157, 250, 63, 170, 106,
206, 107, 124, 212, 45, 111, 107, 9, 219, 200, 177, 0, 240, 143, 156,
44, 207]

A.2.3. **Key Encryption**

Encrypt the CEK with the recipient's public key using the RSAES-
PKCS1-V1_5 algorithm to produce the JWE Encrypted Key. This example
uses the RSA key represented in JSON Web Key [JWK] format below (with
line breaks within values for display purposes only):

     {"kty":"RSA",
      "n":"sXchDaQebHnPiGvyDOAT4saGEUetSyo9MKLOoWFsueri23bOdgWp4Dy1Wl
           UzewbgBHod5pcM9H95GQRV3JDXboIRROSBigeC5yjU1hGzHHyXss8UDpre
           cbAYxknTcQkhslANGRUZmdTOQ5qTRsLAt6BTYuyvVRdhS8exSZEy_c4gs_
           7svlJJQ4H9_NxsiIoLwAEk7-Q3UXERGYw_75IDrGA84-lA_-Ct4eTlXHBI
           Y2EaV7t7LjJaynVJCpkv4LKjTTAumiGUIuQhrNhZLuF_RJLqHpM2kgWFLU
           7-VTdL1VbC2tejvcI2BlMkEpk1BzBZI0KQB0GaDWFLN-aEAw3vRw",
      "e":"AQAB",
      "d":"VFCWOqXr8nvZNyaaJLXdnNPXZKRaWCjkU5Q2egQQpTBMwhprMzWzpR8Sxq
           1OPThh_J6MUD8Z35wky9b8eEO0pwNS8xlh1lOFRRBoNqDIKVOku0aZb-ry
           nq8cxjDTLZQ6Fz7jSjR1Klop-YKaUHc9GsEofQqYruPhzSA-QgajZGPbE_
           0ZaVDJHfyd7UUBUKunFMScbflYAAOYJqVIVwaYR5zWEEceUjNnTNo_CVSj
           -VvXLO5VZfCUAVLgW4dpf1SrtZjSt34YLsRarSb127reG_DUwg9Ch-Kyvj
           T1SkHgUWRVGcyly7uvVGRSDwsXypdrNinPA4jlhoNdizK2zF2CWQ"
     }

     The resulting JWE Encrypted Key value is:

     [80, 104, 72, 58, 11, 130, 236, 139, 132, 189, 255, 205, 61, 86, 151,
     176, 99, 40, 44, 233, 176, 189, 205, 70, 202, 169, 72, 40, 226, 181,
     156, 223, 120, 156, 115, 232, 150, 209, 145, 133, 104, 112, 237, 156,
     116, 250, 65, 102, 212, 210, 103, 240, 177, 61, 93, 40, 71, 231, 223,
     226, 240, 157, 15, 31, 150, 89, 200, 215, 198, 203, 108, 70, 117, 66,
     212, 238, 193, 205, 23, 161, 169, 218, 243, 203, 128, 214, 127, 253,
     215, 139, 43, 17, 135, 103, 179, 220, 28, 2, 212, 206, 131, 158, 128,
     66, 62, 240, 78, 186, 141, 125, 132, 227, 60, 137, 43, 31, 152, 199,
     54, 72, 34, 212, 115, 11, 152, 101, 70, 42, 219, 233, 142, 66, 151,
     250, 126, 146, 141, 216, 190, 73, 50, 177, 146, 5, 52, 247, 28, 197,
     21, 59, 170, 247, 181, 89, 131, 241, 169, 182, 246, 99, 15, 36, 102,
     166, 182, 172, 197, 136, 230, 120, 60, 58, 219, 243, 149, 94, 222,
     150, 154, 194, 110, 227, 225, 112, 39, 89, 233, 112, 207, 211, 241,
     124, 174, 69, 221, 179, 107, 196, 225, 127, 167, 112, 226, 12, 242,
     16, 24, 28, 120, 182, 244, 213, 244, 153, 194, 162, 69, 160, 244,
     248, 63, 165, 141, 4, 207, 249, 193, 79, 131, 0, 169, 233, 127, 167,
     101, 151, 125, 56, 112, 111, 248, 29, 232, 90, 29, 147, 110, 169,
     146, 114, 165, 204, 71, 136, 41, 252]

     Encoding this JWE Encrypted Key as BASE64URL(JWE Encrypted Key) gives
     this value (with line breaks for display purposes only):

       UGhIOguC7IuEvf_NPVaXsGMoLOmwvc1GyqlIKOK1nN94nHPoltGRhWhw7Zx0-kFm
       1NJn8LE9XShH59_i8J0PH5ZZyNfGy2xGdULU7sHNF6Gp2vPLgNZ__deLKxGHZ7Pc
       HALUzoOegEI-8E66jX2E4zyJKx-YxzZIItRzC5hlRirb6Y5Cl_p-ko3YvkkysZIF
       NPccxRU7qve1WYPxqbb2Yw8kZqa2rMWI5ng8OtvzlV7elprCbuPhcCdZ6XDP0_F8
       rkXds2vE4X-ncOIM8hAYHHi29NX0mcKiRaD0-D-ljQTP-cFPgwCp6X-nZZd9OHBv
       -B3oWh2TbqmScqXMR4gp_A

A.2.4. Initialization Vector

Generate a random 128 bit JWE Initialization Vector. In this
example, the value is:

[3, 22, 60, 12, 43, 67, 104, 105, 108, 108, 105, 99, 111, 116, 104,
101]

Encoding this JWE Initialization Vector as BASE64URL(JWE
Initialization Vector) gives this value:

   AxY8DCtDaGlsbGljb3RoZQ

A.2.5. Additional Authenticated Data

Let the Additional Authenticated Data encryption parameter be
ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:

[101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 83, 85, 48, 69,
120, 88, 122, 85, 105, 76, 67, 74, 108, 98, 109, 77, 105, 79, 105,
74, 66, 77, 84, 73, 52, 81, 48, 74, 68, 76, 85, 104, 84, 77, 106, 85,
50, 73, 110, 48]

A.2.6. Content Encryption

Perform authenticated encryption on the Plaintext with the
AES_128_CBC_HMAC_SHA_256 algorithm using the CEK as the encryption
key, the JWE Initialization Vector, and the Additional Authenticated
Data value above. The steps for doing this using the values from
Appendix A.3 are detailed in Appendix B. The resulting Ciphertext
is:

[40, 57, 83, 181, 119, 33, 133, 148, 198, 185, 243, 24, 152, 230, 6,
75, 129, 223, 127, 19, 210, 82, 183, 230, 168, 33, 215, 104, 143,
112, 56, 102]

The resulting Authentication Tag value is:

[246, 17, 244, 190, 4, 95, 98, 3, 231, 0, 115, 157, 242, 203, 100,
191]

Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this
value:

   KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY

Encoding this JWE Authentication Tag as BASE64URL(JWE Authentication
Tag) gives this value:

```
   9hH0vgRfYgPnAHOd8stkvw
```

A.2.7. Complete Representation

Assemble the final representation: The Compact Serialization of this
result is the string BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE Initialization
Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE
Authentication Tag).

The final result in this example (with line breaks for display
purposes only) is:

```
   eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.
   UGhIOguC7IuEvf_NPVaXsGMoLOmwvc1GyqlIKOK1nN94nHPoltGRhWhw7Zx0-kFm
   1NJn8LE9XShH59_i8J0PH5ZZyNfGy2xGdULU7sHNF6Gp2vPLgNZ__deLKxGHZ7Pc
   HALUzoOegEI-8E66jX2E4zyJKx-YxzZIItRzC5hlRirb6Y5Cl_p-ko3YvkkysZIF
   NPccxRU7qve1WYPxqbb2Yw8kZqa2rMWI5ng8OtvzlV7elprCbuPhcCdZ6XDP0_F8
   rkXds2vE4X-ncOIM8hAYHHi29NX0mcKiRaD0-D-ljQTP-cFPgwCp6X-nZZd9OHBv
   -B3oWh2TbqmScqXMR4gp_A.
   AxY8DCtDaGlsbGljb3RoZQ.
   KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY.
   9hH0vgRfYgPnAHOd8stkvw
```

A.2.8. Validation

This example illustrates the process of creating a JWE with RSAES-
PKCS1-V1_5 for key encryption and AES_CBC_HMAC_SHA2 for content
encryption. These results can be used to validate JWE decryption
implementations for these algorithms. Note that since the RSAES-
PKCS1-V1_5 computation includes random values, the encryption results
above will not be completely reproducible. However, since the AES
CBC computation is deterministic, the JWE Encrypted Ciphertext values
will be the same for all encryptions performed using these inputs.

A.3. Example JWE using AES Key Wrap and AES_128_CBC_HMAC_SHA_256

This example encrypts the plaintext "Live long and prosper." to the
recipient using AES Key Wrap for key encryption and
AES_128_CBC_HMAC_SHA_256 for content encryption. The representation
of this plaintext (using JSON array notation) is:

```
[76, 105, 118, 101, 32, 108, 111, 110, 103, 32, 97, 110, 100, 32,
112, 114, 111, 115, 112, 101, 114, 46]
```

[A.3.1](). **JOSE Header**

The following example JWE Protected Header declares that:

o the Content Encryption Key is encrypted to the recipient using the
   AES Key Wrap algorithm with a 128 bit key to produce the JWE
   Encrypted Key and

o authenticated encryption is performed on the Plaintext using the
   AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext and
   the Authentication Tag.

   {"alg":"A128KW","enc":"A128CBC-HS256"}

Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected
Header)) gives this value:

   eyJhbGciOiJBMTI4S1ciLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0

[A.3.2](). **Content Encryption Key (CEK)**

Generate a 256 bit random Content Encryption Key (CEK). In this
example, the value is:

[4, 211, 31, 197, 84, 157, 252, 254, 11, 100, 157, 250, 63, 170, 106,
206, 107, 124, 212, 45, 111, 107, 9, 219, 200, 177, 0, 240, 143, 156,
44, 207]

[A.3.3](). **Key Encryption**

Encrypt the CEK with the shared symmetric key using the AES Key Wrap
algorithm to produce the JWE Encrypted Key. This example uses the
symmetric key represented in JSON Web Key [JWK] format below:

   {"kty":"oct",
    "k":"GawgguFyGrWKav7AX4VKUg"
   }

The resulting JWE Encrypted Key value is:

[232, 160, 123, 211, 183, 76, 245, 132, 200, 128, 123, 75, 190, 216,
22, 67, 201, 138, 193, 186, 9, 91, 122, 31, 246, 90, 28, 139, 57, 3,
76, 124, 193, 11, 98, 37, 173, 61, 104, 57]

Encoding this JWE Encrypted Key as BASE64URL(JWE Encrypted Key) gives
this value:

```
      6KB707dM9YTIgHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ
```

A.3.4. Initialization Vector

Generate a random 128 bit JWE Initialization Vector. In this
example, the value is:

```
[3, 22, 60, 12, 43, 67, 104, 105, 108, 108, 105, 99, 111, 116, 104,
101]
```

Encoding this JWE Initialization Vector as BASE64URL(JWE
Initialization Vector) gives this value:

```
   AxY8DCtDaGlsbGljb3RoZQ
```

A.3.5. Additional Authenticated Data

Let the Additional Authenticated Data encryption parameter be
ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:

```
[101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 66, 77, 84, 73, 52,
83, 49, 99, 105, 76, 67, 74, 108, 98, 109, 77, 105, 79, 105, 74, 66,
77, 84, 73, 52, 81, 48, 74, 68, 76, 85, 104, 84, 77, 106, 85, 50, 73,
110, 48]
```

A.3.6. Content Encryption

Perform authenticated encryption on the Plaintext with the
AES_128_CBC_HMAC_SHA_256 algorithm using the CEK as the encryption
key, the JWE Initialization Vector, and the Additional Authenticated
Data value above. The steps for doing this using the values from
this example are detailed in Appendix B. The resulting Ciphertext
is:

```
[40, 57, 83, 181, 119, 33, 133, 148, 198, 185, 243, 24, 152, 230, 6,
75, 129, 223, 127, 19, 210, 82, 183, 230, 168, 33, 215, 104, 143,
112, 56, 102]
```

The resulting Authentication Tag value is:

```
[83, 73, 191, 98, 104, 205, 211, 128, 201, 189, 199, 133, 32, 38,
194, 85]
```

Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this
value:

```
   KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY
```

   Encoding this JWE Authentication Tag as BASE64URL(JWE Authentication
   Tag) gives this value:

      U0m_YmjN04DJvceFICbCVQ

A.3.7. Complete Representation

   Assemble the final representation: The Compact Serialization of this
   result is the string BASE64URL(UTF8(JWE Protected Header)) || '.' ||
   BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE Initialization
   Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE
   Authentication Tag).

   The final result in this example (with line breaks for display
   purposes only) is:

      eyJhbGciOiJBMTI4S1ciLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.
      6KB707dM9YTIgHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ.
      AxY8DCtDaGlsbGljb3RoZQ.
      KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY.
      U0m_YmjN04DJvceFICbCVQ

A.3.8. Validation

   This example illustrates the process of creating a JWE with AES Key
   Wrap for key encryption and AES GCM for content encryption. These
   results can be used to validate JWE decryption implementations for
   these algorithms. Also, since both the AES Key Wrap and AES GCM
   computations are deterministic, the resulting JWE value will be the
   same for all encryptions performed using these inputs. Since the
   computation is reproducible, these results can also be used to
   validate JWE encryption implementations for these algorithms.

A.4. Example JWE using JWE JSON Serialization

   This section contains an example using the JWE JSON Serialization.
   This example demonstrates the capability for encrypting the same
   plaintext to multiple recipients.

   Two recipients are present in this example. The algorithm and key
   used for the first recipient are the same as that used in
   Appendix A.2. The algorithm and key used for the second recipient
   are the same as that used in Appendix A.3. The resulting JWE
   Encrypted Key values are therefore the same; those computations are
   not repeated here.

   The Plaintext, the Content Encryption Key (CEK), JWE Initialization
   Vector, and JWE Protected Header are shared by all recipients (which

   must be the case, since the Ciphertext and Authentication Tag are
   also shared).

A.4.1. JWE Per-Recipient Unprotected Headers

   The first recipient uses the RSAES-PKCS1-V1_5 algorithm to encrypt
   the Content Encryption Key (CEK). The second uses AES Key Wrap to
   encrypt the CEK. Key ID values are supplied for both keys. The two
   per-recipient header values used to represent these algorithms and
   Key IDs are:

      {"alg":"RSA1_5","kid":"2011-04-29"}

   and

      {"alg":"A128KW","kid":"7"}

A.4.2. JWE Protected Header

   Authenticated encryption is performed on the Plaintext using the
   AES_128_CBC_HMAC_SHA_256 algorithm to produce the common JWE
   Ciphertext and JWE Authentication Tag values. The JWE Protected
   Header value representing this is:

      {"enc":"A128CBC-HS256"}

   Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected
   Header)) gives this value:

      eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0

A.4.3. JWE Unprotected Header

   This JWE uses the "jku" Header Parameter to reference a JWK Set. This
   is represented in the following JWE Unprotected Header value as:

      {"jku":"https://server.example.com/keys.jwks"}

A.4.4. Complete JOSE Header Values

   Combining the per-recipient, protected, and unprotected header values
   supplied, the JOSE Header values used for the first and second
   recipient respectively are:

      {"alg":"RSA1_5",
       "kid":"2011-04-29",
       "enc":"A128CBC-HS256",
       "jku":"https://server.example.com/keys.jwks"}

and

```
{"alg":"A128KW",
 "kid":"7",
 "enc":"A128CBC-HS256",
 "jku":"https://server.example.com/keys.jwks"}
```

A.4.5. Additional Authenticated Data

Let the Additional Authenticated Data encryption parameter be
ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:

[101, 121, 74, 108, 98, 109, 77, 105, 79, 105, 74, 66, 77, 84, 73,
52, 81, 48, 74, 68, 76, 85, 104, 84, 77, 106, 85, 50, 73, 110, 48]

A.4.6. Content Encryption

Perform authenticated encryption on the Plaintext with the
AES_128_CBC_HMAC_SHA_256 algorithm using the CEK as the encryption
key, the JWE Initialization Vector, and the Additional Authenticated
Data value above. The steps for doing this using the values from
Appendix A.3 are detailed in Appendix B. The resulting Ciphertext
is:

[40, 57, 83, 181, 119, 33, 133, 148, 198, 185, 243, 24, 152, 230, 6,
75, 129, 223, 127, 19, 210, 82, 183, 230, 168, 33, 215, 104, 143,
112, 56, 102]

The resulting Authentication Tag value is:

[51, 63, 149, 60, 252, 148, 225, 25, 92, 185, 139, 245, 35, 2, 47,
207]

Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this
value:

KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY

Encoding this JWE Authentication Tag as BASE64URL(JWE Authentication
Tag) gives this value:

Mz-VPPyU4RlcuYv1IwIvzw

A.4.7. Complete JWE JSON Serialization Representation

The complete JSON Web Encryption JSON Serialization for these values
is as follows (with line breaks within values for display purposes
only):

```
   {"protected":
     "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
    "unprotected":
     {"jku":"https://server.example.com/keys.jwks"},
    "recipients":[
     {"header":
       {"alg":"RSA1_5","kid":"2011-04-29"},
      "encrypted_key":
       "UGhIOguC7IuEvf_NPVaXsGMoLOmwvc1GyqlIKOK1nN94nHPoltGRhWhw7Zx0-
        kFm1NJn8LE9XShH59_i8J0PH5ZZyNfGy2xGdULU7sHNF6Gp2vPLgNZ__deLKx
        GHZ7PcHALUzoOegEI-8E66jX2E4zyJKx-YxzZIItRzC5hlRirb6Y5Cl_p-ko3
        YvkkysZIFNPccxRU7qve1WYPxqbb2Yw8kZqa2rMWI5ng8OtvzlV7elprCbuPh
        cCdZ6XDP0_F8rkXds2vE4X-ncOIM8hAYHHi29NX0mcKiRaD0-D-ljQTP-cFPg
        wCp6X-nZZd9OHBv-B3oWh2TbqmScqXMR4gp_A"},
     {"header":
       {"alg":"A128KW","kid":"7"},
      "encrypted_key":
       "6KB707dM9YTIgHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ"}],
    "iv":
     "AxY8DCtDaGlsbGljb3RoZQ",
    "ciphertext":
     "KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY",
    "tag":
     "Mz-VPPyU4RlcuYv1IwIvzw"
   }
```

Appendix B. Example AES_128_CBC_HMAC_SHA_256 Computation

   This example shows the steps in the AES_128_CBC_HMAC_SHA_256
   authenticated encryption computation using the values from the
   example in Appendix A.3. As described where this algorithm is
   defined in Sections 5.2 and 5.2.3 of JWA, the AES_CBC_HMAC_SHA2
   family of algorithms are implemented using Advanced Encryption
   Standard (AES) in Cipher Block Chaining (CBC) mode with PKCS #7
   padding to perform the encryption and an HMAC SHA-2 function to
   perform the integrity calculation - in this case, HMAC SHA-256.

B.1. Extract MAC_KEY and ENC_KEY from Key

   The 256 bit AES_128_CBC_HMAC_SHA_256 key K used in this example
   (using JSON array notation) is:

   [4, 211, 31, 197, 84, 157, 252, 254, 11, 100, 157, 250, 63, 170, 106,
   206, 107, 124, 212, 45, 111, 107, 9, 219, 200, 177, 0, 240, 143, 156,
   44, 207]

   Use the first 128 bits of this key as the HMAC SHA-256 key MAC_KEY,

which is:

[4, 211, 31, 197, 84, 157, 252, 254, 11, 100, 157, 250, 63, 170, 106, 206]

Use the last 128 bits of this key as the AES CBC key ENC_KEY, which is:

[107, 124, 212, 45, 111, 107, 9, 219, 200, 177, 0, 240, 143, 156, 44, 207]

Note that the MAC key comes before the encryption key in the input key K; this is in the opposite order of the algorithm names in the identifiers "AES_128_CBC_HMAC_SHA_256" and "A128CBC-HS256".

B.2. Encrypt Plaintext to Create Ciphertext

Encrypt the Plaintext with AES in Cipher Block Chaining (CBC) mode using PKCS #7 padding using the ENC_KEY above. The Plaintext in this example is:

[76, 105, 118, 101, 32, 108, 111, 110, 103, 32, 97, 110, 100, 32, 112, 114, 111, 115, 112, 101, 114, 46]

The encryption result is as follows, which is the Ciphertext output:

[40, 57, 83, 181, 119, 33, 133, 148, 198, 185, 243, 24, 152, 230, 6, 75, 129, 223, 127, 19, 210, 82, 183, 230, 168, 33, 215, 104, 143, 112, 56, 102]

B.3. 64 Bit Big Endian Representation of AAD Length

The Additional Authenticated Data (AAD) in this example is:

[101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 66, 77, 84, 73, 52, 83, 49, 99, 105, 76, 67, 74, 108, 98, 109, 77, 105, 79, 105, 74, 66, 77, 84, 73, 52, 81, 48, 74, 68, 76, 85, 104, 84, 77, 106, 85, 50, 73, 110, 48]

This AAD is 51 bytes long, which is 408 bits long. The octet string AL, which is the number of bits in AAD expressed as a big endian 64 bit unsigned integer is:

[0, 0, 0, 0, 0, 0, 1, 152]

B.4. **Initialization Vector Value**

   The Initialization Vector value used in this example is:

   [3, 22, 60, 12, 43, 67, 104, 105, 108, 108, 105, 99, 111, 116, 104,
   101]

B.5. **Create Input to HMAC Computation**

   Concatenate the AAD, the Initialization Vector, the Ciphertext, and
   the AL value. The result of this concatenation is:

   [101, 121, 74, 104, 98, 71, 99, 105, 79, 105, 74, 66, 77, 84, 73, 52,
   83, 49, 99, 105, 76, 67, 74, 108, 98, 109, 77, 105, 79, 105, 74, 66,
   77, 84, 73, 52, 81, 48, 74, 68, 76, 85, 104, 84, 77, 106, 85, 50, 73,
   110, 48, 3, 22, 60, 12, 43, 67, 104, 105, 108, 108, 105, 99, 111,
   116, 104, 101, 40, 57, 83, 181, 119, 33, 133, 148, 198, 185, 243, 24,
   152, 230, 6, 75, 129, 223, 127, 19, 210, 82, 183, 230, 168, 33, 215,
   104, 143, 112, 56, 102, 0, 0, 0, 0, 0, 0, 1, 152]

B.6. **Compute HMAC Value**

   Compute the HMAC SHA-256 of the concatenated value above. This
   result M is:

   [83, 73, 191, 98, 104, 205, 211, 128, 201, 189, 199, 133, 32, 38,
   194, 85, 9, 84, 229, 201, 219, 135, 44, 252, 145, 102, 179, 140, 105,
   86, 229, 116]

B.7. **Truncate HMAC Value to Create Authentication Tag**

   Use the first half (128 bits) of the HMAC output M as the
   Authentication Tag output T. This truncated value is:

   [83, 73, 191, 98, 104, 205, 211, 128, 201, 189, 199, 133, 32, 38,
   194, 85]

Appendix C. **Acknowledgements**

   Solutions for encrypting JSON content were also explored by JSON
   Simple Encryption [JSE] and JavaScript Message Security Format
   [I-D.rescorla-jsms], both of which significantly influenced this
   draft. This draft attempts to explicitly reuse as many of the
   relevant concepts from XML Encryption 1.1
   [W3C.REC-xmlenc-core1-20130411] and RFC 5652 [RFC5652] as possible,
   while utilizing simple, compact JSON-based data structures.

Special thanks are due to John Bradley, Eric Rescorla, and Nat
Sakimura for the discussions that helped inform the content of this
specification, to Eric Rescorla and Joe Hildebrand for allowing the
reuse of text from [I-D.rescorla-jsms] in this document, and to Eric
Rescorla for co-authoring many drafts of this specification.

Thanks to Axel Nennker, Emmanuel Raviart, Brian Campbell, and Edmund
Jay for validating the examples in this specification.

This specification is the work of the JOSE Working Group, which
includes dozens of active and dedicated participants. In particular,
the following individuals contributed ideas, feedback, and wording
that influenced this specification:

Richard Barnes, John Bradley, Brian Campbell, Breno de Medeiros, Dick
Hardt, Jeff Hodges, Russ Housley, Edmund Jay, Scott Kelly, Stephen
Kent, James Manger, Matt Miller, Kathleen Moriarty, Tony Nadalin,
Hideki Nara, Axel Nennker, Emmanuel Raviart, Eric Rescorla, Nat
Sakimura, Jim Schaad, Hannes Tschofenig, and Sean Turner.

Jim Schaad and Karen O'Donoghue chaired the JOSE working group and
Sean Turner, Stephen Farrell, and Kathleen Moriarty served as
Security area directors during the creation of this specification.

Appendix D. **Document History**

[[to be removed by the RFC Editor before publication as an RFC]]

-33

o Noted that certificate thumbprints are also sometimes known as
   certificate fingerprints.

o Changed to use the term "authenticated encryption" instead of
   "encryption", where appropriate.

o Acknowledged additional contributors.

-32

o Addressed Gen-ART review comments by Russ Housley.

o Addressed secdir review comments by Scott Kelly, Tero Kivinen, and
   Stephen Kent.

-31

o Updated the reference to draft-mcgrew-aead-aes-cbc-hmac-sha2.

-30

o Added subsection headings within the Overview section for the two
   serializations.

o Added references and cleaned up the reference syntax in a few
   places.

o Applied minor wording changes to the Security Considerations
   section and made other local editorial improvements.

-29

o Replaced the terms JWS Header, JWE Header, and JWT Header with a
   single JOSE Header term defined in the JWS specification. This
   also enabled a single Header Parameter definition to be used and
   reduced other areas of duplication between specifications.

-28

o Specified the use of PKCS #7 padding with AES CBC, rather than
   PKCS #5. (PKCS #7 is a superset of PKCS #5, and is appropriate
   for the 16 octet blocks used by AES CBC.)

o Revised the introduction to the Security Considerations section.
   Also moved a security consideration item here from the JWA draft.

-27

o Described additional security considerations.

o Added the "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) header
   parameter.

-26

o Noted that octet sequences are depicted using JSON array notation.

o Updated references, including to W3C specifications.

-25

o Corrected two external section number references that had changed.

o Corrected a typo in an algorithm name in the prose of an example.

-24

o Corrected complete JSON Serialization example.

o Replaced uses of the term "associated data" wherever it was used
   to refer to a data value with "additional authenticated data",
   since both terms were being used as synonyms, causing confusion.

o Updated the JSON reference to RFC 7159.

o Thanked Eric Rescorla for helping to author of most of the drafts
   of this specification and removed him from the current author
   list.

-23

o Corrected a use of the word "payload" to "plaintext".

-22

o Corrected RFC 2119 terminology usage.

o Replaced references to draft-ietf-json-rfc4627bis with RFC 7158.

-21

o Changed some references from being normative to informative,
   addressing issue #90.

o Applied review comments to the JSON Serialization section,
   addressing issue #178.

-20

o Made terminology definitions more consistent, addressing issue
   #165.

o Restructured the JSON Serialization section to call out the
   parameters used in hanging lists, addressing issue #178.

o Replaced references to RFC 4627 with draft-ietf-json-rfc4627bis,
   addressing issue #90.

-19

o Reordered the key selection parameters.

-18

o Updated the mandatory-to-implement (MTI) language to say that
   applications using this specification need to specify what
   serialization and serialization features are used for that
   application, addressing issue #176.

o Changes to address editorial and minor issues #89, #135, #165,
   #174, #175, #177, #179, and #180.

o Used Header Parameter Description registry field.

-17

o Refined the "typ" and "cty" definitions to always be MIME Media
   Types, with the omission of "application/" prefixes recommended
   for brevity, addressing issue #50.

o Updated the mandatory-to-implement (MTI) language to say that
   general-purpose implementations must implement the single
   recipient case for both serializations whereas special-purpose
   implementations can implement just one serialization if that meets
   the needs of the use cases the implementation is designed for,
   addressing issue #176.

o Explicitly named all the logical components of a JWE and defined
   the processing rules and serializations in terms of those
   components, addressing issues #60, #61, and #62.

o Replaced verbose repetitive phases such as "base64url encode the
   octets of the UTF-8 representation of X" with mathematical
   notation such as "BASE64URL(UTF8(X))".

o Header Parameters and processing rules occurring in both JWS and
   JWE are now referenced in JWS by JWE, rather than duplicated,
   addressing issue #57.

o Terms used in multiple documents are now defined in one place and
   incorporated by reference. Some lightly used or obvious terms
   were also removed. This addresses issue #58.

-16

o Changes to address editorial and minor issues #163, #168, #169,
   #170, #172, and #173.

-15

o Clarified that it is an application decision which recipients'
   encrypted content must successfully validate for the JWE to be

accepted, addressing issue #35.

o Changes to address editorial issues #34, #164, and #169.

-14

o Clarified that the "protected", "unprotected", "header", "iv",
   "tag", and "encrypted_key" parameters are to be omitted in the JWE
   JSON Serialization when their values would be empty. Stated that
   the "recipients" array must always be present.

-13

o Added an "aad" (Additional Authenticated Data) member for the JWE
   JSON Serialization, enabling Additional Authenticated Data to be
   supplied that is not double base64url encoded, addressing issue
   #29.

-12

o Clarified that the "typ" and "cty" header parameters are used in
   an application-specific manner and have no effect upon the JWE
   processing.

o Replaced the MIME types "application/jwe+json" and
   "application/jwe" with "application/jose+json" and
   "application/jose".

o Stated that recipients MUST either reject JWEs with duplicate
   Header Parameter Names or use a JSON parser that returns only the
   lexically last duplicate member name.

o Moved the "epk", "apu", and "apv" Header Parameter definitions to
   be with the algorithm descriptions that use them.

o Added a Serializations section with parallel treatment of the JWE
   Compact Serialization and the JWE JSON Serialization and also
   moved the former Implementation Considerations content there.

o Restored use of the term "AEAD".

o Changed terminology from "block encryption" to "content
   encryption".

-11

o Added Key Identification section.

o Removed the Encrypted Key value from the AAD computation since it
   is already effectively integrity protected by the encryption
   process. The AAD value now only contains the representation of
   the JWE Encrypted Header.

o For the JWE JSON Serialization, enable Header Parameter values to
   be specified in any of three parameters: the "protected" member
   that is integrity protected and shared among all recipients, the
   "unprotected" member that is not integrity protected and shared
   among all recipients, and the "header" member that is not
   integrity protected and specific to a particular recipient. (This
   does not affect the JWE Compact Serialization, in which all Header
   Parameter values are in a single integrity protected JWE Header
   value.)

o Shortened the names "authentication_tag" to "tag" and
   "initialization_vector" to "iv" in the JWE JSON Serialization,
   addressing issue #20.

o Removed "apv" (agreement PartyVInfo) since it is no longer used.

o Removed suggested compact serialization for multiple recipients.

o Changed the MIME type name "application/jwe-js" to
   "application/jwe+json", addressing issue #22.

o Tightened the description of the "crit" (critical) header
   parameter.

-10

o Changed the JWE processing rules for multiple recipients so that a
   single AAD value contains the header parameters and encrypted key
   values for all the recipients, enabling AES GCM to be safely used
   for multiple recipients.

o Added an appendix suggesting a possible compact serialization for
   JWEs with multiple recipients.

-09

o Added JWE JSON Serialization, as specified by
   draft-jones-jose-jwe-json-serialization-04.

o Registered "application/jwe-js" MIME type and "JWE-JS" typ header
   parameter value.

o Defined that the default action for header parameters that are not
   understood is to ignore them unless specifically designated as
   "MUST be understood" or included in the new "crit" (critical)
   header parameter list. This addressed issue #6.

o Corrected "x5c" description. This addressed issue #12.

o Changed from using the term "byte" to "octet" when referring to 8
   bit values.

o Added Key Management Mode definitions to terminology section and
   used the defined terms to provide clearer key management
   instructions. This addressed issue #5.

o Added text about preventing the recipient from behaving as an
   oracle during decryption, especially when using RSAES-PKCS1-V1_5.

o Changed from using the term "Integrity Value" to "Authentication
   Tag".

o Changed member name from "integrity_value" to "authentication_tag"
   in the JWE JSON Serialization.

o Removed Initialization Vector from the AAD value since it is
   already integrity protected by all of the authenticated encryption
   algorithms specified in the JWA specification.

o Replaced "A128CBC+HS256" and "A256CBC+HS512" with "A128CBC-HS256"
   and "A256CBC-HS512". The new algorithms perform the same
   cryptographic computations as [I-D.mcgrew-aead-aes-cbc-hmac-sha2],
   but with the Initialization Vector and Authentication Tag values
   remaining separate from the Ciphertext value in the output
   representation. Also deleted the header parameters "epu"
   (encryption PartyUInfo) and "epv" (encryption PartyVInfo), since
   they are no longer used.

-08

o Replaced uses of the term "AEAD" with "Authenticated Encryption",
   since the term AEAD in the RFC 5116 sense implied the use of a
   particular data representation, rather than just referring to the
   class of algorithms that perform authenticated encryption with
   associated data.

o Applied editorial improvements suggested by Jeff Hodges and Hannes
   Tschofenig. Many of these simplified the terminology used.

o Clarified statements of the form "This header parameter is
   OPTIONAL" to "Use of this header parameter is OPTIONAL".

o Added a Header Parameter Usage Location(s) field to the IANA JSON
   Web Signature and Encryption Header Parameters registry.

o Added seriesInfo information to Internet Draft references.

-07

o Added a data length prefix to PartyUInfo and PartyVInfo values.

o Updated values for example AES CBC calculations.

o Made several local editorial changes to clean up loose ends left
   over from to the decision to only support block encryption methods
   providing integrity. One of these changes was to explicitly state
   that the "enc" (encryption method) algorithm must be an
   Authenticated Encryption algorithm with a specified key length.

-06

o Removed the "int" and "kdf" parameters and defined the new
   composite Authenticated Encryption algorithms "A128CBC+HS256" and
   "A256CBC+HS512" to replace the former uses of AES CBC, which
   required the use of separate integrity and key derivation
   functions.

o Included additional values in the Concat KDF calculation -- the
   desired output size and the algorithm value, and optionally
   PartyUInfo and PartyVInfo values. Added the optional header
   parameters "apu" (agreement PartyUInfo), "apv" (agreement
   PartyVInfo), "epu" (encryption PartyUInfo), and "epv" (encryption
   PartyVInfo). Updated the KDF examples accordingly.

o Promoted Initialization Vector from being a header parameter to
   being a top-level JWE element. This saves approximately 16 bytes
   in the compact serialization, which is a significant savings for
   some use cases. Promoting the Initialization Vector out of the
   header also avoids repeating this shared value in the JSON
   serialization.

o Changed "x5c" (X.509 Certificate Chain) representation from being
   a single string to being an array of strings, each containing a
   single base64 encoded DER certificate value, representing elements
   of the certificate chain.

o   Added an AES Key Wrap example.

o   Reordered the encryption steps so CMK creation is first, when
    required.

o   Correct statements in examples about which algorithms produce
    reproducible results.

-05

o   Support both direct encryption using a shared or agreed upon
    symmetric key, and the use of a shared or agreed upon symmetric
    key to key wrap the CMK.

o   Added statement that "StringOrURI values are compared as case-
    sensitive strings with no transformations or canonicalizations
    applied".

o   Updated open issues.

o   Indented artwork elements to better distinguish them from the body
    text.

-04

o   Refer to the registries as the primary sources of defined values
    and then secondarily reference the sections defining the initial
    contents of the registries.

o   Normatively reference XML Encryption 1.1 for its security
    considerations.

o   Reference draft-jones-jose-jwe-json-serialization instead of
    draft-jones-json-web-encryption-json-serialization.

o   Described additional open issues.

o   Applied editorial suggestions.

-03

o   Added the "kdf" (key derivation function) header parameter to
    provide crypto agility for key derivation. The default KDF
    remains the Concat KDF with the SHA-256 digest function.

o   Reordered encryption steps so that the Encoded JWE Header is
    always created before it is needed as an input to the
    Authenticated Encryption "additional authenticated data"

      parameter.

   o Added the "cty" (content type) header parameter for declaring type
      information about the secured content, as opposed to the "typ"
      (type) header parameter, which declares type information about
      this object.

   o Moved description of how to determine whether a header is for a
      JWS or a JWE from the JWT spec to the JWE spec.

   o Added complete encryption examples for both Authenticated
      Encryption and non-Authenticated Encryption algorithms.

   o Added complete key derivation examples.

   o Added "Collision Resistant Namespace" to the terminology section.

   o Reference ITU.X690.1994 for DER encoding.

   o Added Registry Contents sections to populate registry values.

   o Numerous editorial improvements.

   -02

   o When using Authenticated Encryption algorithms (such as AES GCM),
      use the "additional authenticated data" parameter to provide
      integrity for the header, encrypted key, and ciphertext and use
      the resulting "authentication tag" value as the JWE Authentication
      Tag.

   o Defined KDF output key sizes.

   o Generalized text to allow key agreement to be employed as an
      alternative to key wrapping or key encryption.

   o Changed compression algorithm from gzip to DEFLATE.

   o Clarified that it is an error when a "kid" value is included and
      no matching key is found.

   o Clarified that JWEs with duplicate Header Parameter Names MUST be
      rejected.

   o Clarified the relationship between "typ" header parameter values
      and MIME types.

o Registered application/jwe MIME type and "JWE" typ header
   parameter value.

o Simplified JWK terminology to get replace the "JWK Key Object" and
   "JWK Container Object" terms with simply "JSON Web Key (JWK)" and
   "JSON Web Key Set (JWK Set)" and to eliminate potential confusion
   between single keys and sets of keys. As part of this change, the
   Header Parameter Name for a public key value was changed from
   "jpk" (JSON Public Key) to "jwk" (JSON Web Key).

o Added suggestion on defining additional header parameters such as
   "x5t#S256" in the future for certificate thumbprints using hash
   algorithms other than SHA-1.

o Specify RFC 2818 server identity validation, rather than RFC 6125
   (paralleling the same decision in the OAuth specs).

o Generalized language to refer to Message Authentication Codes
   (MACs) rather than Hash-based Message Authentication Codes (HMACs)
   unless in a context specific to HMAC algorithms.

o Reformatted to give each header parameter its own section heading.

-01

o Added an integrity check for non-Authenticated Encryption
   algorithms.

o Added "jpk" and "x5c" header parameters for including JWK public
   keys and X.509 certificate chains directly in the header.

o Clarified that this specification is defining the JWE Compact
   Serialization. Referenced the new JWE-JS spec, which defines the
   JWE JSON Serialization.

o Added text "New header parameters should be introduced sparingly
   since an implementation that does not understand a parameter MUST
   reject the JWE".

o Clarified that the order of the encryption and decryption steps is
   not significant in cases where there are no dependencies between
   the inputs and outputs of the steps.

o Made other editorial improvements suggested by JOSE working group
   participants.

-00

o Created the initial IETF draft based upon
   draft-jones-json-web-encryption-02 with no normative changes.

o Changed terminology to no longer call both digital signatures and
   HMACs "signatures".

Authors' Addresses

   Michael B. Jones
   Microsoft

   Email: mbj@microsoft.com
   URI: http://self-issued.info/

   Joe Hildebrand
   Cisco Systems, Inc.

   Email: jhildebr@cisco.com