JOSE Working Group	M. Jones
Internet-Draft	Microsoft
Intended status: Standards Track	E. Rescorla
Expires: January 17, 2013	RTFM
	J. Hildebrand
	Cisco
	July 16, 2012

TOC

JSON Web Encryption (JWE)
draft-ietf-jose-json-web-encryption-04

Abstract

JSON Web Encryption (JWE) is a means of representing encrypted content using JavaScript Object Notation (JSON) data structures. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification. Related digital signature and MAC capabilities are described in the separate JSON Web Signature (JWS) specification.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”

This Internet-Draft will expire on January 17, 2013.

Table of Contents

1. Introduction

TOC

JSON Web Encryption (JWE) is a compact encryption format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. It represents this content using JavaScript Object Notation (JSON) **[RFC4627]** based data structures. The JWE cryptographic mechanisms encrypt and provide integrity protection for arbitrary sequences of bytes.

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

1.1. Notational Conventions

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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]**.

2. Terminology

TOC

JSON Web Encryption (JWE)
: A data structure representing an encrypted message. The structure consists of four parts: the JWE Header, the JWE Encrypted Key, the JWE Ciphertext, and the JWE Integrity Value.

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

Ciphertext
: An encrypted representation of the Plaintext.

Content Encryption Key (CEK)
: A symmetric key used to encrypt the Plaintext for the recipient to produce the Ciphertext.

Content Integrity Key (CIK)
: A key used with a MAC function to ensure the integrity of the Ciphertext and the parameters used to create it.

Content Master Key (CMK)
: A key from which the CEK and CIK are derived. When key wrapping or key encryption are employed, the CMK is randomly generated and encrypted to the recipient as the JWE Encrypted Key. When key agreement is employed, the CMK is the result of the key agreement algorithm.

JWE Header
: A string representing a JSON object that describes the encryption operations applied to create the JWE Encrypted Key, the JWE Ciphertext, and the JWE Integrity Value.

JWE Encrypted Key
: When key wrapping or key encryption are employed, the Content Master Key (CMK) is encrypted with the intended recipient's key and the resulting encrypted content is recorded as a byte array, which is referred to as the JWE Encrypted Key. Otherwise, when key agreement is employed, the JWE Encrypted Key is the empty byte array.

JWE Ciphertext
: A byte array containing the Ciphertext.

JWE Integrity Value
: A byte array containing a MAC value that ensures the integrity of the Ciphertext and the parameters used to create it.

Base64url Encoding
: The URL- and filename-safe Base64 encoding described in **RFC 4648** [RFC4648],

Section 5, with the (non URL-safe) '=' padding characters omitted, as permitted by Section 3.2. (See Appendix C of **[JWS]** for notes on implementing base64url encoding without padding.)

Encoded JWE Header
: Base64url encoding of the bytes of the UTF-8 **[RFC3629]** representation of the JWE Header.

Encoded JWE Encrypted Key
: Base64url encoding of the JWE Encrypted Key.

Encoded JWE Ciphertext
: Base64url encoding of the JWE Ciphertext.

Encoded JWE Integrity Value
: Base64url encoding of the JWE Integrity Value.

Header Parameter Name
: The name of a member of the JSON object representing a JWE Header.

Header Parameter Value
: The value of a member of the JSON object representing a JWE Header.

JWE Compact Serialization
: A representation of the JWE as the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by period ('.') characters.

AEAD Algorithm
: An Authenticated Encryption with Associated Data (AEAD) **[RFC5116]** encryption algorithm is one that provides an integrated content integrity check. AES Galois/Counter Mode (GCM) is one such algorithm.

Collision Resistant Namespace
: A namespace that allows names to be allocated in a manner such that they are highly unlikely to collide with other names. For instance, collision resistance can be achieved through administrative delegation of portions of the namespace or through use of collision-resistant name allocation functions. Examples of Collision Resistant Namespaces include: Domain Names, Object Identifiers (OIDs) as defined in the ITU-T X.660 and X.670 Recommendation series, and Universally Unique IDentifiers (UUIDs) **[RFC4122]**. When using an administratively delegated namespace, the definer of a name needs to take reasonable precautions to ensure they are in control of the portion of the namespace they use to define the name.

StringOrURI
: A JSON string value, with the additional requirement that while arbitrary string values MAY be used, any value containing a ":" character MUST be a URI **[RFC3986]**.

---

3. JSON Web Encryption (JWE) Overview

TOC

JWE represents encrypted content using JSON data structures and base64url encoding. The representation consists of four parts: the JWE Header, the JWE Encrypted Key, the JWE Ciphertext, and the JWE Integrity Value. In the Compact Serialization, the four parts are base64url-encoded for transmission, and represented as the concatenation of the encoded strings in that order, with the four strings being separated by period ('.') characters. (A JSON Serialization for this information is defined in the separate JSON Web Encryption JSON Serialization (JWE-JS) **[JWE-JS]** specification.)

JWE utilizes encryption to ensure the confidentiality of the Plaintext. JWE adds a content integrity check if not provided by the underlying encryption algorithm.

3.1. Example JWE with an Integrated Integrity Check

TOC

This example encrypts the plaintext "Live long and prosper." to the recipient using RSAES OAEP and AES GCM. The AES GCM algorithm has an integrated integrity check.

The following example JWE Header declares that:

- the Content Master Key is encrypted to the recipient using the RSAES OAEP algorithm to produce the JWE Encrypted Key,

- the Plaintext is encrypted using the AES GCM algorithm with a 256 bit key to produce the Ciphertext, and
- the 96 bit Initialization Vector (IV) with the base64url encoding `48V1_ALb6US04U3b` was used.

```
{"alg":"RSA-OAEP","enc":"A256GCM","iv":"48V1_ALb6US04U3b"}
```

Base64url encoding the bytes of the UTF-8 representation of the JWE Header yields this Encoded JWE Header value (with line breaks for display purposes only):

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00iLCJpdiI6IjQ4VjFfQUxi
NlVTMDRVM2IifQ
```

The remaining steps to finish creating this JWE are:

- Generate a random Content Master Key (CMK)
- Encrypt the CMK with the recipient's public key using the RSAES OAEP algorithm to produce the JWE Encrypted Key
- Base64url encode the JWE Encrypted Key to produce the Encoded JWE Encrypted Key
- Concatenate the Encoded JWE Header value, a period character ('.'), and the Encoded JWE Encrypted Key to create the "additional authenticated data" parameter for the AES GCM algorithm.
- Encrypt the Plaintext with AES GCM, using the IV, the CMK as the encryption key, and the "additional authenticated data" value above, requesting a 128 bit "authentication tag" output
- Base64url encode the resulting Ciphertext to create the Encoded JWE Ciphertext
- Base64url encode the resulting "authentication tag" to create the Encoded JWE Integrity Value
- Assemble the final representation: The Compact Serialization of this result is the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by three period ('.') characters.

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

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00iLCJpdiI6IjQ4VjFfQUxi
NlVTMDRVM2IifQ.
jvwoyhWxOMboB5cxX6ncAi7Wp3Q5FKRtlmIx35pfR9HpEa6Oy-iEpxEqM30W3YcR
Q8WU9ouRoO5jd6tfdcpX-2X-OteHw4dnMXdMLjHGGx86LMDeFRAN2KGz7EGPJiva
w0yM80fzT3zY0PKrIvU5ml1M5szqUnX4Jw0-PNcIM_j-L5YkLhv3Yk04XCwTJwxN
NmXCflYAQO9f00Aa213TJJr6dbHV6I642FwU-EWvtEfN3evgX3EFIVYSnT3HCHkA
AIdBQ9ykD-abRzVA_dGp_yJAZQcrZuNTqzThd_22YMPhIpzTygfC_4k7qqxI6t7L
e_l5_o-taUG7vaNAl5FjEQ.
_e21tGGhac_peEFkLXr2dMPUZiUkrw.
YbZSeHCNDZBqAdzpROlyiw
```

See **Appendix A.1** for the complete details of computing this JWE.

3.2. Example JWE with a Separate Integrity Check

TOC

This example encrypts the plaintext "Now is the time for all good men to come to the aid of their country." to the recipient using RSAES-PKCS1-V1_5 and AES CBC. AES CBC does not have an integrated integrity check, so a separate integrity check calculation is performed using HMAC SHA-256, with separate encryption and integrity keys being derived from a master key using the Concat KDF with the SHA-256 digest function.

The following example JWE Header (with line breaks for display purposes only) declares that:

- the Content Master Key is encrypted to the recipient using the RSAES-PKCS1-

V1_5 algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES CBC algorithm with a 128 bit key to produce the Ciphertext,
- the JWE Integrity Value safeguarding the integrity of the Ciphertext and the parameters used to create it was computed with the HMAC SHA-256 algorithm, and
- the 128 bit Initialization Vector (IV) with the base64url encoding `AxY8DCtDaGlsbGljb3RoZQ` was used.

```
{"alg":"RSA1_5","enc":"A128CBC","int":"HS256","iv":"AxY8DCtDaGls
bGljb3RoZQ"}
```

Base64url encoding the bytes of the UTF-8 representation of the JWE Header yields this Encoded JWE Header value (with line breaks for display purposes only):

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ
```

The remaining steps to finish creating this JWE are like the previous example, but with an additional step to compute the separate integrity value:

- Generate a random Content Master Key (CMK)
- Encrypt the CMK with the recipient's public key using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key
- Base64url encode the JWE Encrypted Key to produce the Encoded JWE Encrypted Key
- Use the Concat key derivation function to derive Content Encryption Key (CEK) and Content Integrity Key (CIK) values from the CMK
- Encrypt the Plaintext with AES CBC using the CEK and IV to produce the Ciphertext
- Base64url encode the resulting Ciphertext to create the Encoded JWE Ciphertext
- Concatenate the Encoded JWE Header value, a period character ('.'), the Encoded JWE Encrypted Key, a second period character, and the Encoded JWE Ciphertext to create the value to integrity protect
- Compute the HMAC SHA-256 of this value using the CIK to create the JWE Integrity Value
- Base64url encode the resulting JWE Integrity Value to create the Encoded JWE Integrity Value
- Assemble the final representation: The Compact Serialization of this result is the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by three period ('.') characters.

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

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ.
IPI_z172hSWHMFgED8EG9DM6hIXU_6NaO1DImCn0vNeuoBq847Sl6qw_GHSYHJUQ
XtXJq7S_CxWVrI82wjrOyaQca5tLZRZc45BfKHeqByThKI261QevEK56SyAwwXfK
KZjSvkQ5dwTFSgfy76rMSUvVynHYEhdCatBF9HWTAiXPx7hgZixG1FeP_QCmOylz
2VClVyYFCbjKREOwBFf-puNYfO75S3LNlJUtTsGGQL2oTKpMsEiUTdefkje91VX9
h8g7908lFsggbjV7NicJsufuXxnTj1fcWIrRDeNIOmakiPEODi0gTSz0ou-W-LWK
-3T1zYlOIiIKBjsExQKZ-w.
_Z_djlIoC4MDSCKireWS2beti4Q6iSG2UjFujQvdz-_PQdUcFNkOulegD6BgjgdF
LjeB4HHOO7UHvP8PEDu0a0sA2a_-CI0w2YQQ2QQe35M.
c41k4T4eAgCCt63m8ZNmiOinMciFFypOFpvid7i6D0k
```

See **Appendix A.2** for the complete details of computing this JWE.

4. JWE Header

The members of the JSON object represented by the JWE Header describe the encryption applied to the Plaintext and optionally additional properties of the JWE. The Header Parameter Names within this object MUST be unique; JWEs with duplicate Header Parameter Names MUST be rejected. Implementations MUST understand the entire contents of the header; otherwise, the JWE MUST be rejected.

There are two ways of distinguishing whether a header is a JWS Header or a JWE Header. The first is by examining the `alg` (algorithm) header value. If the value represents a digital signature or MAC algorithm, or is the value `none`, it is for a JWS; if it represents an encryption or key agreement algorithm, it is for a JWE. A second method is determining whether an `enc` (encryption method) member exists. If the `enc` member exists, it is a JWE; otherwise, it is a JWS. Both methods will yield the same result.

There are three classes of Header Parameter Names: Reserved Header Parameter Names, Public Header Parameter Names, and Private Header Parameter Names.

4.1. Reserved Header Parameter Names

The following header parameter names are reserved with meanings as defined below. All the names are short because a core goal of JWE is for the representations to be compact.

Additional reserved header parameter names MAY be defined via the IANA JSON Web Signature and Encryption Header Parameters registry **[JWS]**. 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

The `alg` (algorithm) header parameter identifies the cryptographic algorithm used to encrypt or reach agreement upon the Content Master Key (CMK). The algorithm specified by the `alg` value MUST be supported by the implementation and there MUST be a key for use with that algorithm associated with the intended recipient or the JWE MUST be rejected. `alg` values SHOULD either be registered in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]** or be a URI that contains a Collision Resistant Namespace. The `alg` value is a case sensitive string containing a StringOrURI value. This header parameter is REQUIRED.

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

4.1.2. "enc" (Encryption Method) Header Parameter

The `enc` (encryption method) header parameter identifies the symmetric encryption algorithm used to encrypt the Plaintext to produce the Ciphertext. The algorithm specified by the `enc` value MUST be supported by the implementation or the JWE MUST be rejected. `enc` values SHOULD either be registered in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]** or be a URI that contains a Collision Resistant Namespace. The `enc` value is a case sensitive string containing a StringOrURI value. This header parameter is REQUIRED.

A list of defined `enc` values can be found in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]**; the initial contents of this registry is the values defined in Section 4.2 of the JSON Web Algorithms (JWA) **[JWA]** specification.

4.1.3. "int" (Integrity Algorithm) Header Parameter

TOC

The `int` (integrity algorithm) header parameter identifies the cryptographic algorithm used to safeguard the integrity of the Ciphertext and the parameters used to create it. The `int` parameter uses the MAC subset of the algorithm values used by the JWS `alg` parameter. `int` values SHOULD either be registered in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]** or be a URI that contains a Collision Resistant Namespace. The `int` value is a case sensitive string containing a StringOrURI value. This header parameter is REQUIRED when an AEAD algorithm is not used to encrypt the Plaintext and MUST NOT be present when an AEAD algorithm is used.

A list of defined `int` values can be found in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]**; the initial contents of this registry is the values defined in Section 4.3 of the JSON Web Algorithms (JWA) **[JWA]** specification.

4.1.4. "kdf" (Key Derivation Function) Header Parameter

TOC

The `kdf` (key derivation function) header parameter identifies the cryptographic algorithm used to derive the CEK and CIK from the CMK. `kdf` values SHOULD either be registered in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]** or be a URI that contains a Collision Resistant Namespace. The `kdf` value is a case sensitive string containing a StringOrURI value. This header parameter is OPTIONAL when an AEAD algorithm is not used to encrypt the Plaintext and MUST NOT be present when an AEAD algorithm is used.

When an AEAD algorithm is not used and no `kdf` header parameter is present, the `CS256` KDF **[JWA]** SHALL be used.

A list of defined `kdf` values can be found in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]**; the initial contents of this registry is the values defined in Section 4.4 of the JSON Web Algorithms (JWA) **[JWA]** specification.

4.1.5. "iv" (Initialization Vector) Header Parameter

TOC

The `iv` (initialization vector) value for algorithms requiring it, represented as a base64url encoded string. This header parameter is OPTIONAL, although its use is REQUIRED with some `enc` algorithms.

4.1.6. "epk" (Ephemeral Public Key) Header Parameter

TOC

The `epk` (ephemeral public key) value created by the originator for the use in key agreement algorithms. This key is represented as a JSON Web Key **[JWK]** value. This header parameter is OPTIONAL, although its use is REQUIRED with some `alg` algorithms.

4.1.7. "zip" (Compression Algorithm) Header Parameter

TOC

The `zip` (compression algorithm) applied to the Plaintext before encryption, if any. If present, the value of the `zip` header parameter MUST be the case sensitive string "DEF". Compression is performed with the DEFLATE **[RFC1951]** algorithm. If no `zip` parameter is present, no compression is applied to the Plaintext before encryption. This header parameter is OPTIONAL.

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

TOC

The jku (JWK Set URL) header parameter is a URI **[RFC3986]** that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the key used to encrypt the JWE; this can be used to determine the private key needed to decrypt the JWE. The keys MUST be encoded as a JSON Web Key Set (JWK Set) **[JWK]**. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS **[RFC2818]** **[RFC5246]**; the identity of the server MUST be validated, as per Section 3.1 of HTTP Over TLS **[RFC2818]**. This header parameter is OPTIONAL.

TOC

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

The jwk (JSON Web Key) header parameter is a public key that corresponds to the key used to encrypt the JWE; this can be used to determine the private key needed to decrypt the JWE. This key is represented as a JSON Web Key **[JWK]**. This header parameter is OPTIONAL.

TOC

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

The x5u (X.509 URL) header parameter is a URI **[RFC3986]** that refers to a resource for the X.509 public key certificate or certificate chain **[RFC5280]** corresponding to the key used to encrypt the JWE; this can be used to determine the private key needed to decrypt the JWE. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to **RFC 5280** [RFC5280] in PEM encoded form **[RFC1421]**. The certificate containing the public key of the entity that encrypted the JWE MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS **[RFC2818]** **[RFC5246]**; the identity of the server MUST be validated, as per Section 3.1 of HTTP Over TLS **[RFC2818]**. This header parameter is OPTIONAL.

TOC

4.1.11. "x5t" (X.509 Certificate Thumbprint) Header Parameter

The x5t (X.509 Certificate Thumbprint) header parameter provides a base64url encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of the X.509 certificate **[RFC5280]** corresponding to the key used to encrypt the JWE; this can be used to determine the private key needed to decrypt the JWE. This header parameter is OPTIONAL.

If, in the future, certificate thumbprints need to be computed using hash functions other than SHA-1, it is suggested that additional related header parameters be defined for that purpose. For example, it is suggested that a new x5t#S256 (X.509 Certificate Thumbprint using SHA-256) header parameter could be defined by registering it in the IANA JSON Web Signature and Encryption Header Parameters registry **[JWS]**.

TOC

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

The x5c (X.509 Certificate Chain) header parameter contains the X.509 public key certificate or certificate chain **[RFC5280]** corresponding to the key used to encrypt the JWE; this can be used to determine the private key needed to decrypt the JWE. The certificate or certificate chain is represented as an array of certificate values. Each value is a base64 encoded (**[RFC4648]** Section 4 - not base64url encoded) DER **[ITU.X690.1994]** PKIX certificate value. The certificate containing the public key of the entity that encrypted the JWE MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The recipient MUST verify the certificate chain according to **[RFC5280]** and reject the JWE if any validation failure occurs. This header parameter is OPTIONAL.

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

4.1.13. "kid" (Key ID) Header Parameter

TOC

The `kid` (key ID) header parameter is a hint indicating which key was used to encrypt the JWE; 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 recipients. Should the recipient be unable to locate a key corresponding to the `kid` value, they SHOULD treat that condition as an error. The interpretation of the `kid` value is unspecified. Its value MUST be a string. This header parameter is OPTIONAL.

When used with a JWK, the `kid` value MAY be used to match a JWK `kid` parameter value.

4.1.14. "typ" (Type) Header Parameter

TOC

The `typ` (type) header parameter is used to declare the type of this object. The type value `JWE` MAY be used to indicate that this object is a JWE. The `typ` value is a case sensitive string. This header parameter is OPTIONAL.

MIME Media Type **[RFC2046]** values MAY be used as `typ` values.

`typ` values SHOULD either be registered in the IANA JSON Web Signature and Encryption Type Values registry **[JWS]** or be a URI that contains a Collision Resistant Namespace.

4.1.15. "cty" (Content Type) Header Parameter

TOC

The `cty` (content type) header parameter is used to declare the type of the encrypted content (the Plaintext). The `cty` value is a case sensitive string. This header parameter is OPTIONAL.

The values used for the `cty` header parameter come from the same value space as the `typ` header parameter, with the same rules applying.

4.2. Public Header Parameter Names

TOC

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 **[JWS]** or be a URI that contains a Collision Resistant Namespace. 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

TOC

A producer and consumer of a JWE may agree to any header parameter name that is not a Reserved Name **Section 4.1** or a Public Name **Section 4.2**. Unlike Public Names, these private names are subject to collision and should be used with caution.

5. Message Encryption

TOC

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. When key wrapping or key encryption are employed, generate a random Content Master Key (CMK). See **RFC 4086** [RFC4086] for considerations on generating random values. Otherwise, when key agreement is employed, use the key agreement algorithm to compute the value of the Content Master Key (CMK). The CMK MUST have a length equal to that of the larger of the required encryption and integrity keys.
2. When key wrapping or key encryption are employed, encrypt the CMK for the recipient (see **Section 7**) and let the result be the JWE Encrypted Key. Otherwise, when key agreement is employed, let the JWE Encrypted Key be an empty byte array.
3. Base64url encode the JWE Encrypted Key to create the Encoded JWE Encrypted Key.
4. Generate a random Initialization Vector (IV) of the correct size for the algorithm (if required for the algorithm).
5. If not using an AEAD algorithm, run the key derivation algorithm specified by the `kdf` header parameter to generate the Content Encryption Key (CEK) and the Content Integrity Key (CIK); otherwise (when using an AEAD algorithm), set the CEK to be the CMK.
6. Compress the Plaintext if a `zip` parameter was included.
7. Serialize the (compressed) Plaintext into a byte sequence M.
8. Create a JWE Header containing the encryption parameters used. Note that white space is explicitly allowed in the representation and no canonicalization need be performed before encoding.
9. Base64url encode the bytes of the UTF-8 representation of the JWE Header to create the Encoded JWE Header.
10. Encrypt M using the CEK and IV to form the byte sequence C. If an AEAD algorithm is used, use the bytes of the ASCII representation of the concatenation of the Encoded JWE Header, a period ('.') character, and the Encoded JWE Encrypted Key as the "additional authenticated data" parameter value for the encryption.
11. Base64url encode C to create the Encoded JWE Ciphertext.
12. If not using an AEAD algorithm, run the integrity algorithm (see **Section 8**) using the CIK to compute the JWE Integrity Value; otherwise (when using an AEAD algorithm), set the JWE Integrity Value to be the "authentication tag" value produced by the AEAD algorithm.
13. Base64url encode the JWE Integrity Value to create the Encoded JWE Integrity Value.
14. The four encoded parts, taken together, are the result.
15. The Compact Serialization of this result is the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by period ('.') characters.

---

6. Message Decryption

TOC

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 JWE MUST be rejected.

1. Determine the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value values contained in the JWE. When using the Compact Serialization, these four values are represented in that order, separated by period characters.
2. The Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value MUST be successfully base64url decoded following the restriction that no padding characters have been used.
3. The resulting JWE Header MUST be completely valid JSON syntax conforming to **RFC 4627** [RFC4627].
4. The resulting JWE Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported.
5. Verify that the JWE Header references a key known to the recipient.
6. When key wrapping or key encryption are employed, decrypt the JWE Encrypted Key to produce the Content Master Key (CMK). Otherwise, when key agreement

is employed, use the key agreement algorithm to compute the value of the Content Master Key (CMK). The CMK MUST have a length equal to that of the larger of the required encryption and integrity keys.
7. If not using an AEAD algorithm, run the key derivation algorithm specified by the `kdf` header parameter to generate the Content Encryption Key (CEK) and the Content Integrity Key (CIK); otherwise (when using an AEAD algorithm), set the CEK to be the CMK.
8. Decrypt the binary representation of the JWE Ciphertext using the CEK and IV. If an AEAD algorithm is used, use the bytes of the ASCII representation of the concatenation of the Encoded JWE Header, a period ('.') character, and the Encoded JWE Encrypted Key as the "additional authenticated data" parameter value for the decryption.
9. If not using an AEAD algorithm, run the integrity algorithm (see **Section 8**) using the CIK to compute an integrity value for the input received. This computed value MUST match the received JWE Integrity Value; otherwise (when using an AEAD algorithm), the received JWE Integrity Value MUST match the "authentication tag" value produced by the AEAD algorithm.
10. Uncompress the result of the previous step, if a `zip` parameter was included.
11. Output the resulting Plaintext.

---

7. CMK Encryption

TOC

JWE supports two forms of Content Master Key (CMK) encryption:

- Asymmetric encryption under the recipient's public key.
- Symmetric encryption under a key shared between the sender and receiver.

See the algorithms registered for `enc` usage in the IANA JSON Web Signature and Encryption Algorithms registry **[JWA]** and Section 4.1 of the JSON Web Algorithms (JWA) **[JWA]** specification for lists of encryption algorithms that can be used for CMK encryption.

---

8. Integrity Value Calculation

TOC

When a non-AEAD algorithm is used (an algorithm without an integrated content check), JWE adds an explicit integrity check value to the representation. This value is computed in the manner described in the JSON Web Signature (JWS) **[JWS]** specification, with these modifications:

- The algorithm used is taken from the `int` (integrity algorithm) header parameter rather than the `alg` header parameter.
- The algorithm MUST be a MAC algorithm (such as HMAC SHA-256).
- The JWS Secured Input used is the bytes of the ASCII representation of the concatenation of the Encoded JWE Header, a period ('.') character, the Encoded JWE Encrypted Key, a period ('.') character, and the Encoded JWE Ciphertext.
- The CIK is used as the MAC key.

The computed JWS Signature value is the resulting integrity value.

---

9. Encrypting JWEs with Cryptographic Algorithms

TOC

JWE uses cryptographic algorithms to encrypt the Plaintext and the Content Encryption Key (CMK) and to provide integrity protection for the JWE Header, JWE Encrypted Key, and JWE Ciphertext. The JSON Web Algorithms (JWA) **[JWA]** specification specifies a set of cryptographic algorithms and identifiers to be used with this specification and defines registries for additional such algorithms. Specifically, Section 4.1 specifies a set of `alg` (algorithm) header parameter values, Section 4.2 specifies a set of `enc` (encryption method) header parameter values, Section 4.3 specifies a set of `int` (integrity algorithm) header parameter values, and Section 4.4 specifies a set of `kdf` (key derivation function) header parameter values intended for use this specification. It also describes the semantics and operations that are specific to these algorithms and algorithm families.

Public keys employed for encryption can be identified using the Header Parameter methods described in **Section 4.1** or can be distributed using methods that are outside the scope of this specification.

10. IANA Considerations

TOC

10.1. Registration of JWE Header Parameter Names

TOC

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

10.1.1. Registry Contents

TOC

- Header Parameter Name: `alg`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.1** of [[this document]]

- Header Parameter Name: `enc`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.2** of [[this document]]

- Header Parameter Name: `int`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.3** of [[this document]]

- Header Parameter Name: `kdf`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.4** of [[this document]]

- Header Parameter Name: `iv`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.5** of [[this document]]

- Header Parameter Name: `epk`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.6** of [[this document]]

- Header Parameter Name: `zip`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.7** of [[this document]]

- Header Parameter Name: `jku`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.8** of [[this document]]

- Header Parameter Name: `jwk`
- Change Controller: IETF
- Specification document(s): **Section 4.1.9** of [[this document]]

- Header Parameter Name: `x5u`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.10** of [[this document]]

- Header Parameter Name: `x5t`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.11** of [[this document]]

- Header Parameter Name: `x5c`
- Change Controller: IETF

- Specification Document(s): **Section 4.1.12** of [[this document]]

- Header Parameter Name: `kid`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.13** of [[this document]]

- Header Parameter Name: `typ`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.14** of [[this document]]

- Header Parameter Name: `cty`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.15** of [[this document]]

10.2. JSON Web Signature and Encryption Type Values Registration

TOC

10.2.1. Registry Contents

TOC

This specification registers the `JWE` type value in the IANA JSON Web Signature and Encryption Type Values registry **[JWS]**:

- "typ" Header Parameter Value: `JWE`
- Abbreviation for MIME Type: application/jwe
- Change Controller: IETF
- Specification Document(s): **Section 4.1.14** of [[this document]]

10.3. Media Type Registration

TOC

10.3.1. Registry Contents

TOC

This specification registers the `application/jwe` Media Type **[RFC2046]** in the MIME Media Type registry **[RFC4288]** to indicate that the content is a JWE using the Compact Serialization.

- Type Name: application
- Subtype Name: jwe
- Required Parameters: n/a
- Optional Parameters: n/a
- Encoding considerations: JWE values are encoded as a series of base64url encoded values (some of which may be the empty string) separated by period ('.') characters
- Security Considerations: See the Security Considerations section of this document
- Interoperability Considerations: n/a
- Published Specification: [[this document]]
- Applications that use this media type: OpenID Connect and other applications using encrypted JWTs
- Additional Information: Magic number(s): n/a, File extension(s): n/a, Macintosh file type code(s): n/a
- Person & email address to contact for further information: Michael B. Jones, mbj@microsoft.com
- Intended Usage: COMMON
- Restrictions on Usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change Controller: IETF

11. Security Considerations

TOC

All of the security issues faced by any cryptographic application must be faced by a JWS/JWE/JWK agent. Among these issues are protecting the user's private key, preventing various attacks, and helping the user avoid mistakes such as inadvertently encrypting a message for the wrong recipient. The entire list of security considerations is beyond the scope of this document, but some significant concerns are listed here.

All the security considerations in the JWS specification also apply to this specification. Likewise, all the security considerations in **XML Encryption 1.1** [W3C.CR-xmlenc-core1-20120313] also apply to JWE, other than those that are XML specific.

12. Open Issues

TOC

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

The following items remain to be considered or done in this draft:

- Should we define an optional nonce and/or timestamp header parameter? (Use of a nonce is an effective countermeasure to some kinds of attacks.)
- When doing key agreement, do we want to also use a separate CMK and encrypt the CMK with the agreed upon key or just use the agreed upon key directly as the CMK? Or support both? Having a CMK would have value in the multiple recipients case, as it would allow multiple recipients to share the same ciphertext even when key agreement is used, but it seems that it's just extra overhead in the single recipient case. (Also see the related open issue about performing symmetric encryption directly with a shared key, without using a CMK.)
- Do we want to consolidate the combination of the `enc`, `int`, and `kdf` parameters into a single new `enc` parameter defining composite AEAD algorithms? For instance, we might define a composite algorithm A128CBC with HS256 and CS256 and another composite algorithm A256CBC with HS512 and CS512. A symmetry argument for doing this is that the `int` and `kdf` parameters are not used with AEAD algorithms. An argument against it is that in some cases, integrity is not needed because it's provided by other means, and so having the flexibility to not use an `int` algorithm or key derivation with a non-AEAD `enc` algorithm could be useful.

13. References

TOC

13.1. Normative References

TOC

[ITU.X690.1994] International Telecommunications Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)," ITU-T Recommendation X.690, 1994.

[JWA] **Jones, M.**, "**JSON Web Algorithms (JWA)**," July 2012.

[JWK] **Jones, M.**, "**JSON Web Key (JWK)**," July 2012.

[JWS] **Jones, M.**, **Bradley, J.**, and **N. Sakimura**, "**JSON Web Signature (JWS)**," July 2012.

[RFC1421] **Linn, J.**, "**Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures**," RFC 1421, February 1993 (**TXT**).

[RFC1951] **Deutsch, P.**, "**DEFLATE Compressed Data Format Specification version 1.3**," RFC 1951, May 1996 (**TXT**, **PS**, **PDF**).

[RFC2046] **Freed, N.** and **N. Borenstein**, "**Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types**," RFC 2046, November 1996 (**TXT**).

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

[RFC2818] Rescorla, E., "**HTTP Over TLS**," RFC 2818, May 2000 (**TXT**).

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

[RFC3986] **Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, "**Uniform Resource Identifier (URI): Generic Syntax**," STD 66, RFC 3986, January 2005 (**TXT**, **HTML**, **XML**).

[RFC4086]	Eastlake, D., Schiller, J., and S. Crocker, "**Randomness Requirements for Security**," BCP 106, RFC 4086, June 2005 (**TXT**).
[RFC4288]	Freed, N. and J. Klensin, "**Media Type Specifications and Registration Procedures**," BCP 13, RFC 4288, December 2005 (**TXT**).
[RFC4627]	Crockford, D., "**The application/json Media Type for JavaScript Object Notation (JSON)**," RFC 4627, July 2006 (**TXT**).
[RFC4648]	Josefsson, S., "**The Base16, Base32, and Base64 Data Encodings**," RFC 4648, October 2006 (**TXT**).
[RFC5116]	McGrew, D., "**An Interface and Algorithms for Authenticated Encryption**," RFC 5116, January 2008 (**TXT**).
[RFC5246]	Dierks, T. and E. Rescorla, "**The Transport Layer Security (TLS) Protocol Version 1.2**," RFC 5246, August 2008 (**TXT**).
[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 (**TXT**).
[W3C.CR-xmlenc-core1-20120313]	Eastlake, D., Reagle, J., Hirsch, F., and T. Roessler, "**XML Encryption Syntax and Processing Version 1.1**," World Wide Web Consortium CR CR-xmlenc-core1-20120313, March 2012 (**HTML**).

TOC

13.2. Informative References

[I-D.rescorla-jsms]	Rescorla, E. and J. Hildebrand, "**JavaScript Message Security Format**," draft-rescorla-jsms-00 (work in progress), March 2011 (**TXT**).
[JSE]	Bradley, J. and N. Sakimura (editor), "**JSON Simple Encryption**," September 2010.
[JWE-JS]	**Jones, M.**, "**JSON Web Encryption JSON Serialization (JWE-JS)**," July 2012.
[RFC4122]	**Leach, P.**, **Mealling, M.**, and **R. Salz**, "**A Universally Unique IDentifier (UUID) URN Namespace**," RFC 4122, July 2005 (**TXT**, **HTML**, **XML**).
[RFC5652]	Housley, R., "**Cryptographic Message Syntax (CMS)**," STD 70, RFC 5652, September 2009 (**TXT**).

TOC

Appendix A. JWE Examples

This section provides examples of JWE computations.

TOC

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

This example encrypts the plaintext "Live long and prosper." to the recipient using RSAES OAEP and AES GCM. The AES GCM algorithm has an integrated integrity check. The representation of this plaintext is:

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

TOC

A.1.1. JWE Header

The following example JWE Header declares that:

- the Content Master Key is encrypted to the recipient using the RSAES OAEP algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES GCM algorithm with a 256 bit key to produce the Ciphertext, and
- the 96 bit Initialization Vector (IV) [227, 197, 117, 252, 2, 219, 233, 68, 180, 225, 77, 219] with the base64url encoding `48V1_ALb6US04U3b` was used.

```
{"alg":"RSA-OAEP","enc":"A256GCM","iv":"48V1_ALb6US04U3b"}
```

TOC

A.1.2. Encoded JWE Header

Base64url encoding the bytes of the UTF-8 representation of the JWE Header yields this Encoded JWE Header value (with line breaks for display purposes only):

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00iLCJpdiI6IjQ4VjFfQUxi
NlVTMDRVM2IifQ
```

A.1.3. Content Master Key (CMK)

TOC

Generate a random Content Master Key (CMK). In this example, the key value 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.1.4. Key Encryption

TOC

Encrypt the CMK with the recipient's public key using the RSAES OAEP algorithm to produce the JWE Encrypted Key. In this example, the RSA key parameters are:

Parameter Name	Value
Modulus	[161, 168, 84, 34, 133, 176, 208, 173, 46, 176, 163, 110, 57, 30, 135, 227, 9, 31, 226, 128, 84, 92, 116, 241, 70, 248, 27, 227, 193, 62, 5, 91, 241, 145, 224, 205, 141, 176, 184, 133, 239, 43, 81, 103, 9, 161, 153, 157, 179, 104, 123, 51, 189, 34, 152, 69, 97, 69, 78, 93, 140, 131, 87, 182, 169, 101, 92, 142, 3, 22, 167, 8, 212, 56, 35, 79, 210, 222, 192, 208, 252, 49, 109, 138, 173, 253, 210, 166, 201, 63, 102, 74, 5, 158, 41, 90, 144, 108, 160, 79, 10, 89, 222, 231, 172, 31, 227, 197, 0, 19, 72, 81, 138, 78, 136, 221, 121, 118, 196, 17, 146, 10, 244, 188, 72, 113, 55, 221, 162, 217, 171, 27, 57, 233, 210, 101, 236, 154, 199, 56, 138, 239, 101, 48, 198, 186, 202, 160, 76, 111, 234, 71, 57, 183, 5, 211, 171, 136, 126, 64, 40, 75, 58, 89, 244, 254, 107, 84, 103, 7, 236, 69, 163, 18, 180, 251, 58, 153, 46, 151, 174, 12, 103, 197, 181, 161, 162, 55, 250, 235, 123, 110, 17, 11, 158, 24, 47, 133, 8, 199, 235, 107, 126, 130, 246, 73, 195, 20, 108, 202, 176, 214, 187, 45, 146, 182, 118, 54, 32, 200, 61, 201, 71, 243, 1, 255, 131, 84, 37, 111, 211, 168, 228, 45, 192, 118, 27, 197, 235, 232, 36, 10, 230, 248, 190, 82, 182, 140, 35, 204, 108, 190, 253, 186, 186, 27]
Exponent	[1, 0, 1]
Private Exponent	[144, 183, 109, 34, 62, 134, 108, 57, 44, 252, 10, 66, 73, 54, 16, 181, 233, 92, 54, 219, 101, 42, 35, 178, 63, 51, 43, 92, 119, 136, 251, 41, 53, 23, 191, 164, 164, 60, 88, 227, 229, 152, 228, 213, 149, 228, 169, 237, 104, 71, 151, 75, 88, 252, 216, 77, 251, 231, 28, 97, 88, 193, 215, 202, 248, 216, 121, 195, 211, 245, 250, 112, 71, 243, 61, 129, 95, 39, 244, 122, 225, 217, 169, 211, 165, 48, 253, 220, 59, 122, 219, 42, 86, 223, 32, 236, 39, 48, 103, 78, 122, 216, 187, 88, 176, 89, 24, 1, 42, 177, 24, 99, 142, 170, 1, 146, 43, 3, 108, 64, 194, 121, 182, 95, 187, 134, 71, 88, 96, 134, 74, 131, 167, 69, 106, 143, 121, 27, 72, 44, 245, 95, 39, 194, 179, 175, 203, 122, 16, 112, 183, 17, 200, 202, 31, 17, 138, 156, 184, 210, 157, 184, 154, 131, 128, 110, 12, 85, 195, 122, 241, 79, 251, 229, 183, 117, 21, 123, 133, 142, 220, 153, 9, 59, 57, 105, 81, 255, 138, 77, 82, 54, 62, 216, 38, 249, 208, 17, 197, 49, 45, 19, 232, 157, 251, 131, 137, 175, 72, 126, 43, 229, 69, 179, 117, 82, 157, 213, 83, 35, 57, 210, 197, 252, 171, 143, 194, 11, 47, 163, 6, 253, 75, 252, 96, 11, 187, 84, 130, 210, 7, 121, 78, 91, 79, 57, 251, 138, 132, 220, 60, 224, 173, 56, 224, 201]

The resulting JWE Encrypted Key value is:

[142, 252, 40, 202, 21, 177, 56, 198, 232, 7, 151, 49, 95, 169, 220, 2, 46, 214, 167, 116, 57, 20, 164, 109, 150, 98, 49, 223, 154, 95, 71, 209, 233, 17, 174, 142, 203, 232, 132, 167, 17, 42, 51, 125, 22, 221, 135, 17, 67, 197, 148, 246, 139, 145, 160, 238, 99, 119, 171, 95, 117, 202, 87, 251, 101, 254, 58, 215, 135, 195, 135, 103, 49, 119, 76, 46, 49, 198, 27, 31, 58, 44, 192, 222, 21, 16, 13, 216, 161, 179, 236, 65, 143, 38, 43, 218, 195, 76, 140, 243, 71, 243, 79, 124, 216, 208, 242, 171, 34, 245, 57, 154, 93, 76, 230, 204, 234, 82, 117, 248, 39, 13, 62, 60, 215, 8, 51, 248, 254, 47, 150, 36, 46, 27, 247, 98, 77, 56, 92, 44, 19, 39, 12, 77, 54, 101, 194,

126, 86, 0, 64, 239, 95, 211, 64, 26, 219, 93, 211, 36, 154, 250, 117, 177, 213, 232, 142, 184, 216, 92, 20, 248, 69, 175, 180, 71, 205, 221, 235, 224, 95, 113, 5, 33, 86, 18, 157, 61, 199, 8, 121, 0, 0, 135, 65, 67, 220, 164, 15, 230, 155, 71, 53, 64, 253, 209, 169, 255, 34, 64, 101, 7, 43, 102, 227, 83, 171, 52, 225, 119, 253, 182, 96, 195, 225, 34, 156, 211, 202, 7, 194, 255, 137, 59, 170, 172, 72, 234, 222, 203, 123, 249, 121, 254, 143, 173, 105, 65, 187, 189, 163, 64, 151, 145, 99, 17]

TOC

A.1.5. Encoded JWE Encrypted Key

Base64url encode the JWE Encrypted Key to produce the Encoded JWE Encrypted Key. This result (with line breaks for display purposes only) is:

```
jvwoyhWxOMboB5cxX6ncAi7Wp3Q5FKRtlmIx35pfR9HpEa6Oy-iEpxEqM30W3YcR
Q8WU9ouRoO5jd6tfdcpX-2X-OteHw4dnMXdMLjHGGx86LMDeFRAN2KGz7EGPJiva
w0yM80fzT3zY0PKrIvU5ml1M5szqUnX4Jw0-PNcIM_j-L5YkLhv3Yk04XCwTJwxN
NmXCflYAQO9f00Aa213TJJr6dbHV6I642FwU-EWvtEfN3evgX3EFIVYSnT3HCHkA
AIdBQ9ykD-abRzVA_dGp_yJAZQcrZuNTqzThd_22YMPhIpzTygfC_4k7qqxI6t7L
e_l5_o-taUG7vaNAl5FjEQ
```

TOC

A.1.6. "Additional Authenticated Data" Parameter

Concatenate the Encoded JWE Header value, a period character ('.'), and the Encoded JWE Encrypted Key to create the "additional authenticated data" parameter for the AES GCM algorithm. This result (with line breaks for display purposes only) is:

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00iLCJpdiI6IjQ4VjFfQUxi
NlVTMDRVM2IifQ.
jvwoyhWxOMboB5cxX6ncAi7Wp3Q5FKRtlmIx35pfR9HpEa6Oy-iEpxEqM30W3YcR
Q8WU9ouRoO5jd6tfdcpX-2X-OteHw4dnMXdMLjHGGx86LMDeFRAN2KGz7EGPJiva
w0yM80fzT3zY0PKrIvU5ml1M5szqUnX4Jw0-PNcIM_j-L5YkLhv3Yk04XCwTJwxN
NmXCflYAQO9f00Aa213TJJr6dbHV6I642FwU-EWvtEfN3evgX3EFIVYSnT3HCHkA
AIdBQ9ykD-abRzVA_dGp_yJAZQcrZuNTqzThd_22YMPhIpzTygfC_4k7qqxI6t7L
e_l5_o-taUG7vaNAl5FjEQ
```

The representation of 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, 76, 67, 74, 112, 100, 105, 73, 54, 73, 106, 81, 52, 86, 106, 70, 102, 81, 85, 120, 105, 78, 108, 86, 84, 77, 68, 82, 86, 77, 50, 73, 105, 102, 81, 46, 106, 118, 119, 111, 121, 104, 87, 120, 79, 77, 98, 111, 66, 53, 99, 120, 88, 54, 110, 99, 65, 105, 55, 87, 112, 51, 81, 53, 70, 75, 82, 116, 108, 109, 73, 120, 51, 53, 112, 102, 82, 57, 72, 112, 69, 97, 54, 79, 121, 45, 105, 69, 112, 120, 69, 113, 77, 51, 48, 87, 51, 89, 99, 82, 81, 56, 87, 85, 57, 111, 117, 82, 111, 79, 53, 106, 100, 54, 116, 102, 100, 99, 112, 88, 45, 50, 88, 45, 79, 116, 101, 72, 119, 52, 100, 110, 77, 88, 100, 77, 76, 106, 72, 71, 71, 120, 56, 54, 76, 77, 68, 101, 70, 82, 65, 78, 50, 75, 71, 122, 55, 69, 71, 80, 74, 105, 118, 97, 119, 48, 121, 77, 56, 48, 102, 122, 84, 51, 122, 89, 48, 80, 75, 114, 73, 118, 85, 53, 109, 108, 49, 77, 53, 115, 122, 113, 85, 110, 88, 52, 74, 119, 48, 45, 80, 78, 99, 73, 77, 95, 106, 45, 76, 53, 89, 107, 76, 104, 118, 51, 89, 107, 48, 52, 88, 67, 119, 84, 74, 119, 120, 78, 78, 109, 88, 67, 102, 108, 89, 65, 81, 79, 57, 102, 48, 48, 65, 97, 50, 49, 51, 84, 74, 74, 114, 54, 100, 98, 72, 86, 54, 73, 54, 52, 50, 70, 119, 85, 45, 69, 87, 118, 116, 69, 102, 78, 51, 101, 118, 103, 88, 51, 69, 70, 73, 86, 89, 83, 110, 84, 51, 72, 67, 72, 107, 65, 65, 73, 100, 66, 81, 57, 121, 107, 68, 45, 97, 98, 82, 122, 86, 65, 95, 100, 71, 112, 95, 121, 74, 65, 90, 81, 99, 114, 90, 117, 78, 84, 113, 122, 84, 104, 100, 95, 50, 50, 89, 77, 80, 104, 73, 112, 122, 84, 121, 103, 102, 67, 95, 52, 107, 55, 113, 113, 120, 73, 54, 116, 55, 76, 101, 95, 108, 53, 95, 111, 45, 116, 97, 85, 71, 55, 118, 97, 78, 65, 108, 53, 70, 106, 69, 81]

TOC

A.1.7. Plaintext Encryption

TOC

Encrypt the Plaintext with AES GCM, using the IV, the CMK as the encryption key, and the "additional authenticated data" value above, requesting a 128 bit "authentication tag" output. The resulting Ciphertext is:

[253, 237, 181, 180, 97, 161, 105, 207, 233, 120, 65, 100, 45, 122, 246, 116, 195, 212, 102, 37, 36, 175]

The resulting "authentication tag" value is:

[97, 182, 82, 120, 112, 141, 13, 144, 106, 1, 220, 233, 68, 233, 114, 139]

A.1.8. Encoded JWE Ciphertext

TOC

Base64url encode the resulting Ciphertext to create the Encoded JWE Ciphertext. This result is:

```
_e21tGGhac_peEFkLXr2dMPUZiUkrw
```

A.1.9. Encoded JWE Integrity Value

TOC

Base64url encode the resulting "authentication tag" to create the Encoded JWE Integrity Value. This result is:

```
YbZSeHCNDZBqAdzpROlyiw
```

A.1.10. Complete Representation

TOC

Assemble the final representation: The Compact Serialization of this result is the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by three period ('.') characters.

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

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00iLCJpdiI6IjQ4VjFfQUxi
NlVTMDRVM2IifQ.
jvwoyhWxOMboB5cxX6ncAi7Wp3Q5FKRtlmIx35pfR9HpEa6Oy-iEpxEqM30W3YcR
Q8WU9ouRoO5jd6tfdcpX-2X-OteHw4dnMXdMLjHGGx86LMDeFRAN2KGz7EGPJiva
w0yM80fzT3zY0PKrIvU5ml1M5szqUnX4Jw0-PNcIM_j-L5YkLhv3Yk04XCwTJwxN
NmXCflYAQO9f00Aa213TJJr6dbHV6I642FwU-EWvtEfN3evgX3EFIVYSnT3HCHkA
AIdBQ9ykD-abRzVA_dGp_yJAZQcrZuNTqzThd_22YMPhIpzTygfC_4k7qqxI6t7L
e_l5_o-taUG7vaNAl5FjEQ.
_e21tGGhac_peEFkLXr2dMPUZiUkrw.
YbZSeHCNDZBqAdzpROlyiw
```

A.1.11. Validation

TOC

This example illustrates the process of creating a JWE with an AEAD algorithm. These results can be used to validate JWE decryption implementations for these algorithms. However, note that since the RSAES OAEP computation includes random values, the results above will not be repeatable.

A.2. Example JWE using RSAES-PKCS1-V1_5 and AES CBC

TOC

This example encrypts the plaintext "Now is the time for all good men to come to the aid of their country." to the recipient using RSAES-PKCS1-V1_5 and AES CBC. AES CBC does not have an integrated integrity check, so a separate integrity check calculation is performed using HMAC SHA-256, with separate encryption and integrity keys being derived from a master key using the Concat KDF with the SHA-256 digest function. The representation of this plaintext is:

[78, 111, 119, 32, 105, 115, 32, 116, 104, 101, 32, 116, 105, 109, 101, 32, 102, 111, 114, 32, 97, 108, 108, 32, 103, 111, 111, 100, 32, 109, 101, 110, 32, 116, 111, 32, 99, 111, 109, 101, 32, 116, 111, 32, 116, 104, 101, 32, 97, 105, 100, 32, 111, 102, 32, 116, 104, 101, 105, 114, 32, 99, 111, 117, 110, 116, 114, 121, 46]

A.2.1. JWE Header

TOC

The following example JWE Header (with line breaks for display purposes only) declares that:

- the Content Master Key is encrypted to the recipient using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES CBC algorithm with a 128 bit key to produce the Ciphertext,
- the JWE Integrity Value safeguarding the integrity of the Ciphertext and the parameters used to create it was computed with the HMAC SHA-256 algorithm, and
- the 128 bit Initialization Vector (IV) [3, 22, 60, 12, 43, 67, 104, 105, 108, 108, 105, 99, 111, 116, 104, 101] with the base64url encoding `AxY8DCtDaGlsbGljb3RoZQ` was used.

```
{"alg":"RSA1_5","enc":"A128CBC","int":"HS256","iv":"AxY8DCtDaGls
bGljb3RoZQ"}
```

A.2.2. Encoded JWE Header

TOC

Base64url encoding the bytes of the UTF-8 representation of the JWE Header yields this Encoded JWE Header value (with line breaks for display purposes only):

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ
```

A.2.3. Content Master Key (CMK)

TOC

Generate a random Content Master Key (CMK). 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.2.4. Key Encryption

TOC

Encrypt the CMK with the recipient's public key using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key. In this example, the RSA key parameters are:

Parameter Name	Value
Modulus	[177, 119, 33, 13, 164, 30, 108, 121, 207, 136, 107, 242, 12, 224, 19, 226, 198, 134, 17, 71, 173, 75, 42, 61, 48, 162, 206, 161, 97, 108, 185, 234, 226, 219, 118, 206, 118, 5, 169, 224, 60, 181, 90, 85, 51, 123, 6, 224, 4, 122, 29, 230, 151, 12, 244, 127, 121, 25, 4, 85, 220, 144, 215, 110, 130, 17, 68, 228, 129, 138, 7, 130, 231, 40, 212, 214, 17, 179, 28, 124, 151, 178, 207, 20, 14, 154, 222, 113, 176, 24, 198, 73, 211, 113, 9, 33, 178, 80, 13, 25, 21, 25, 153, 212, 206, 67, 154, 147, 70, 194, 192, 183, 160, 83, 98, 236, 175, 85, 23, 97, 75, 199, 177, 73, 145, 50, 253, 206, 32, 179, 254, 236, 190, 82, 73, 67, 129, 253, 252, 220, 108, 136, 138, 11, 192, 1, 36, 239, 228, 55, 81, 113, 17, 25, 140, 63, 239, 146, 3, 172, 96, 60, 227, 233, 64, 255, 224, 173, 225, 228, 229, 92, 112, 72, 99, 97, 26, 87, 187, 123, 46, 50, 90, 202, 117, 73, 10, 153, 47, 224, 178, 163, 77, 48, 46, 154, 33, 148, 34, 228, 33, 172, 216, 89, 46, 225, 127, 68, 146, 234, 30, 147, 54, 146, 5, 133, 45, 78, 254, 85, 55, 75, 213, 86, 194, 218, 215, 163, 189, 194, 54, 6, 83, 36, 18, 153, 53, 7, 48, 89, 35, 66, 144, 7, 65, 154, 13, 97, 75, 55, 230, 132, 3, 13, 239, 71]
Exponent	[1, 0, 1]
Private Exponent	[84, 80, 150, 58, 165, 235, 242, 123, 217, 55, 38, 154, 36, 181, 221, 156, 211, 215, 100, 164, 90, 88, 40, 228, 83, 148, 54, 122, 4, 16, 165, 48, 76, 194, 26, 107, 51, 53, 179, 165, 31, 18, 198, 173, 78, 61, 56, 97, 252, 158, 140, 80, 63, 25, 223, 156, 36, 203, 214, 252, 120, 67, 180, 167, 3, 82, 243, 25, 97, 214, 83, 133, 69, 16, 104, 54, 160, 200, 41, 83, 164, 187, 70, 153, 111, 234, 242, 158, 175, 28, 198, 48, 211, 45, 148, 58, 23, 62, 227, 74, 52, 117, 42, 90, 41, 249, 130, 154, 80, 119, 61, 26, 193, 40, 125, 10, 152, 174, 227, 225, 205, 32, 62, 66, 6, 163, 100, 99, 219, 19, 253, 25, 105, 80, 201, 29, 252, 157, 237, 69, 1, 80, 171, 167, 20, 196, 156, 109, 249, 88, 0, 3, 152, 38, 165, 72, 87, 6, 152, 71, 156, 214, 16, 71, 30, 82, 51, 103, 76, 218, 63, 9, 84, 163, 249, 91, 215, 44, 238, 85, 101, 240, 148, 1, 82, 224, 91, 135, 105, 127, 84, 171, 181, 152, 210, 183, 126, 24, 46, 196, 90, 173, 38, 245, 219, 186, 222, 27, 240, 212, 194, 15, 66, 135, 226, 178, 190, 52, 245, 74, 65, 224, 81, 100, 85, 25, 204, 165, 203, 187, 175, 84, 100, 82, 15, 11, 23, 202, 151, 107, 54, 41, 207, 3, 136, 229, 134, 131, 93, 139, 50, 182, 204, 93, 130, 89]

The resulting JWE Encrypted Key value is:

[32, 242, 63, 207, 94, 246, 133, 37, 135, 48, 88, 4, 15, 193, 6, 244, 51, 58, 132, 133, 212, 255, 163, 90, 59, 80, 200, 152, 41, 244, 188, 215, 174, 160, 26, 188, 227, 180, 165, 234, 172, 63, 24, 116, 152, 28, 149, 16, 94, 213, 201, 171, 180, 191, 11, 21, 149, 172, 143, 54, 194, 58, 206, 201, 164, 28, 107, 155, 75, 101, 22, 92, 227, 144, 95, 40, 119, 170, 7, 36, 225, 40, 141, 186, 213, 7, 175, 16, 174, 122, 75, 32, 48, 193, 119, 202, 41, 152, 210, 190, 68, 57, 119, 4, 197, 74, 7, 242, 239, 170, 204, 73, 75, 213, 202, 113, 216, 18, 23, 66, 106, 208, 69, 244, 117, 147, 2, 37, 207, 199, 184, 96, 102, 44, 70, 212, 87, 143, 253, 0, 166, 59, 41, 115, 217, 80, 165, 87, 38, 5, 9, 184, 202, 68, 67, 176, 4, 87, 254, 166, 227, 88, 124, 238, 249, 75, 114, 205, 148, 149, 45, 78, 193, 134, 64, 189, 168, 76, 170, 76, 176, 72, 148, 77, 215, 159, 146, 55, 189, 213, 85, 253, 135, 200, 59, 247, 79, 37, 22, 200, 32, 110, 53, 123, 54, 39, 9, 178, 231, 238, 95, 25, 211, 143, 87, 220, 88, 138, 209, 13, 227, 72, 58, 102, 164, 136, 241, 14, 14, 45, 32, 77, 44, 244, 162, 239, 150, 248, 181, 138, 251, 116, 245, 205, 137, 78, 34, 34, 10, 6, 59, 4, 197, 2, 153, 251]

A.2.5. Encoded JWE Encrypted Key

TOC

Base64url encode the JWE Encrypted Key to produce the Encoded JWE Encrypted Key. This result (with line breaks for display purposes only) is:

```
IPI_z172hSWHMFgED8EG9DM6hIXU_6NaO1DImCn0vNeuoBq847Sl6qw_GHSYHJUQ
XtXJq7S_CxWVrI82wjrOyaQca5tLZRZc45BfKHeqByThKI261QevEK56SyAwwXfK
KZjSvkQ5dwTFSgfy76rMSUvVynHYEhdCatBF9HWTAiXPx7hgZixG1FeP_QCmOylz
2VClVyYFCbjKREOwBFf-puNYfO75S3LNlJUtTsGGQL2oTKpMsEiUTdefkje91VX9
h8g7908lFsggbjV7NicJsufuXxnTj1fcWIrRDeNIOmakiPEODi0gTSz0ou-W-LWK
-3T1zYlOIiIKBjsExQKZ-w
```

A.2.6. Key Derivation

TOC

Use the Concat key derivation function to derive Content Encryption Key (CEK) and Content Integrity Key (CIK) values from the CMK. The details of this derivation are shown in **Appendix A.3**. The resulting CEK value is:

[249, 255, 87, 218, 224, 223, 221, 53, 204, 121, 166, 130, 195, 184, 50, 69]

The resulting CIK value is:

[218, 209, 130, 50, 169, 45, 70, 214, 29, 187, 123, 20, 3, 158, 111, 122, 182, 94, 57, 133, 245, 76, 97, 44, 193, 80, 81, 246, 115, 177, 225, 159]

A.2.7. Plaintext Encryption

TOC

Encrypt the Plaintext with AES CBC using the CEK and IV to produce the Ciphertext. The resulting Ciphertext is:

[253, 159, 221, 142, 82, 40, 11, 131, 3, 72, 34, 162, 173, 229, 146, 217, 183, 173, 139, 132, 58, 137, 33, 182, 82, 49, 110, 141, 11, 221, 207, 239, 207, 65, 213, 28, 20, 217, 14, 186, 87, 160, 15, 160, 96, 142, 7, 69, 46, 55, 129, 224, 113, 206, 59, 181, 7, 188, 255, 15, 16, 59, 180, 107, 75, 0, 217, 175, 254, 8, 141, 48, 217, 132, 16, 217, 4, 30, 223, 147]

A.2.8. Encoded JWE Ciphertext

TOC

Base64url encode the resulting Ciphertext to create the Encoded JWE Ciphertext. This result (with line breaks for display purposes only) is:

```
_Z_djlIoC4MDSCKireWS2beti4Q6iSG2UjFujQvdz-_PQdUcFNkOulegD6BgjgdF
LjeB4HHOO7UHvP8PEDu0a0sA2a_-CI0w2YQQ2QQe35M
```

A.2.9. Secured Input Value

TOC

Concatenate the Encoded JWE Header value, a period character ('.'), the Encoded JWE Encrypted Key, a second period character, and the Encoded JWE Ciphertext to create the value to integrity protect. This result (with line breaks for display purposes only) is:

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ.
IPI_z172hSWHMFgED8EG9DM6hIXU_6NaO1DImCn0vNeuoBq847Sl6qw_GHSYHJUQ
XtXJq7S_CxWVrI82wjrOyaQca5tLZRZc45BfKHeqByThKI261QevEK56SyAwwXfK
KZjSvkQ5dwTFSgfy76rMSUvVynHYEhdCatBF9HWTAiXPx7hgZixG1FeP_QCmOylz
2VClVyYFCbjKREOwBFf-puNYfO75S3LNlJUtTsGGQL2oTKpMsEiUTdefkje91VX9
h8g7908lFsggbjV7NicJsufuXxnTj1fcWIrRDeNIOmakiPEODi0gTSz0ou-W-LWK
-3T1zYlOIiIKBjsExQKZ-w.
_Z_djlIoC4MDSCKireWS2beti4Q6iSG2UjFujQvdz-_PQdUcFNkOulegD6BgjgdF
LjeB4HHOO7UHvP8PEDu0a0sA2a_-CI0w2YQQ2QQe35M
```

The representation of 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, 73, 105, 119, 105, 97, 87, 53, 48, 73, 106, 111, 105, 83, 70, 77, 121, 78, 84, 89, 105, 76, 67, 74, 112, 100, 105, 73, 54, 73, 107, 70, 52, 87, 84, 104, 69, 81, 51, 82, 69, 89, 85, 100, 115, 99, 50, 74, 72, 98, 71, 112, 105, 77, 49, 74, 118, 87, 108, 69, 105, 102, 81, 46, 73, 80, 73, 95, 122, 49, 55, 50, 104, 83, 87, 72, 77, 70, 103, 69, 68, 56, 69, 71, 57, 68, 77, 54, 104, 73, 88, 85, 95, 54, 78, 97,

79, 49, 68, 73, 109, 67, 110, 48, 118, 78, 101, 117, 111, 66, 113, 56, 52, 55, 83, 108, 54, 113, 119, 95, 71, 72, 83, 89, 72, 74, 85, 81, 88, 116, 88, 74, 113, 55, 83, 95, 67, 120, 87, 86, 114, 73, 56, 50, 119, 106, 114, 79, 121, 97, 81, 99, 97, 53, 116, 76, 90, 82, 90, 99, 52, 53, 66, 102, 75, 72, 101, 113, 66, 121, 84, 104, 75, 73, 50, 54, 49, 81, 101, 118, 69, 75, 53, 54, 83, 121, 65, 119, 119, 88, 102, 75, 75, 90, 106, 83, 118, 107, 81, 53, 100, 119, 84, 70, 83, 103, 102, 121, 55, 54, 114, 77, 83, 85, 118, 86, 121, 110, 72, 89, 69, 104, 100, 67, 97, 116, 66, 70, 57, 72, 87, 84, 65, 105, 88, 80, 120, 55, 104, 103, 90, 105, 120, 71, 49, 70, 101, 80, 95, 81, 67, 109, 79, 121, 108, 122, 50, 86, 67, 108, 86, 121, 89, 70, 67, 98, 106, 75, 82, 69, 79, 119, 66, 70, 102, 45, 112, 117, 78, 89, 102, 79, 55, 53, 83, 51, 76, 78, 108, 74, 85, 116, 84, 115, 71, 71, 81, 76, 50, 111, 84, 75, 112, 77, 115, 69, 105, 85, 84, 100, 101, 102, 107, 106, 101, 57, 49, 86, 88, 57, 104, 56, 103, 55, 57, 48, 56, 108, 70, 115, 103, 103, 98, 106, 86, 55, 78, 105, 99, 74, 115, 117, 102, 117, 88, 120, 110, 84, 106, 49, 102, 99, 87, 73, 114, 82, 68, 101, 78, 73, 79, 109, 97, 107, 105, 80, 69, 79, 68, 105, 48, 103, 84, 83, 122, 48, 111, 117, 45, 87, 45, 76, 87, 75, 45, 51, 84, 49, 122, 89, 108, 79, 73, 105, 73, 75, 66, 106, 115, 69, 120, 81, 75, 90, 45, 119, 46, 95, 90, 95, 100, 106, 108, 73, 111, 67, 52, 77, 68, 83, 67, 75, 105, 114, 101, 87, 83, 50, 98, 101, 116, 105, 52, 81, 54, 105, 83, 71, 50, 85, 106, 70, 117, 106, 81, 118, 100, 122, 45, 95, 80, 81, 100, 85, 99, 70, 78, 107, 79, 117, 108, 101, 103, 68, 54, 66, 103, 106, 103, 100, 70, 76, 106, 101, 66, 52, 72, 72, 79, 79, 55, 85, 72, 118, 80, 56, 80, 69, 68, 117, 48, 97, 48, 115, 65, 50, 97, 95, 45, 67, 73, 48, 119, 50, 89, 81, 81, 50, 81, 81, 101, 51, 53, 77]

A.2.10. JWE Integrity Value

TOC

Compute the HMAC SHA-256 of this value using the CIK to create the JWE Integrity Value. This result is:

[115, 141, 100, 225, 62, 30, 2, 0, 130, 183, 173, 230, 241, 147, 102, 136, 232, 167, 49, 200, 133, 23, 42, 78, 22, 155, 226, 119, 184, 186, 15, 73]

A.2.11. Encoded JWE Integrity Value

TOC

Base64url encode the resulting JWE Integrity Value to create the Encoded JWE Integrity Value. This result is:

```
c41k4T4eAgCCt63m8ZNmiOinMciFFypOFpvid7i6D0k
```

A.2.12. Complete Representation

TOC

Assemble the final representation: The Compact Serialization of this result is the concatenation of the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value in that order, with the four strings being separated by three period ('.') characters.

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

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ.
IPI_z172hSWHMFgED8EG9DM6hIXU_6NaO1DImCn0vNeuoBq847Sl6qw_GHSYHJUQ
XtXJq7S_CxWVrI82wjrOyaQca5tLZRZc45BfKHeqByThKI261QevEK56SyAwwXfK
KZjSvkQ5dwTFSgfy76rMSUvVynHYEhdCatBF9HWTAiXPx7hgZixG1FeP_QCmOylz
2VClVyYFCbjKREOwBFf-puNYfO75S3LNlJUtTsGGQL2oTKpMsEiUTdefkje91VX9
h8g7908lFsggbjV7NicJsufuXxnTj1fcWIrRDeNIOmakiPEODi0gTSz0ou-W-LWK
-3T1zYlOIiIKBjsExQKZ-w.
_Z_djlIoC4MDSCKireWS2beti4Q6iSG2UjFujQvdz-_PQdUcFNkOulegD6BgjgdF
LjeB4HHOO7UHvP8PEDu0a0sA2a_-CI0w2YQQ2QQe35M.
c41k4T4eAgCCt63m8ZNmiOinMciFFypOFpvid7i6D0k
```

TOC

A.2.13. Validation

This example illustrates the process of creating a JWE with a non-AEAD algorithm. These results can be used to validate JWE decryption implementations for these algorithms. Since all the algorithms used in this example produce deterministic results, the results above should be repeatable.

TOC

A.3. Example Key Derivation with Outputs <= Hash Size

This example uses the Concat KDF to derive the Content Encryption Key (CEK) and Content Integrity Key (CIK) from the Content Master Key (CMK) in the manner described in Section 4.12 of **[JWA]**. In this example, a 256 bit CMK is used to derive a 128 bit CEK and a 256 bit CIK.

The CMK 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]

TOC

A.3.1. CEK Generation

When deriving the CEK from the CMK, the ASCII label "Encryption" ([69, 110, 99, 114, 121, 112, 116, 105, 111, 110]) is used. The input to the first hash round is the concatenation of the big endian number 1 ([0, 0, 0, 1]), the CMK, and the label. Thus the round 1 hash input is:

[0, 0, 0, 1, 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, 69, 110, 99, 114, 121, 112, 116, 105, 111, 110]

The SHA-256 hash of this value, which is the round 1 hash output, is:

[249, 255, 87, 218, 224, 223, 221, 53, 204, 121, 166, 130, 195, 184, 50, 69, 11, 237, 202, 71, 10, 96, 59, 199, 140, 88, 126, 147, 146, 113, 222, 41]

Given that 128 bits are needed for the CEK and the hash has produced 256 bits, the CEK value is the first 128 bits of that value:

[249, 255, 87, 218, 224, 223, 221, 53, 204, 121, 166, 130, 195, 184, 50, 69]

TOC

A.3.2. CIK Generation

When deriving the CIK from the CMK, the ASCII label "Integrity" ([73, 110, 116, 101, 103, 114, 105, 116, 121]) is used. The input to the first hash round is the concatenation of the big endian number 1 ([0, 0, 0, 1]), the CMK, and the label. Thus the round 1 hash input is:

[0, 0, 0, 1, 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, 73, 110, 116, 101, 103, 114, 105, 116, 121]

The SHA-256 hash of this value, which is the round 1 hash output, is:

[218, 209, 130, 50, 169, 45, 70, 214, 29, 187, 123, 20, 3, 158, 111, 122, 182, 94, 57, 133, 245, 76, 97, 44, 193, 80, 81, 246, 115, 177, 225, 159]

Given that 256 bits are needed for the CIK and the hash has produced 256 bits, the CIK value is that same value:

[218, 209, 130, 50, 169, 45, 70, 214, 29, 187, 123, 20, 3, 158, 111, 122, 182, 94, 57, 133, 245, 76, 97, 44, 193, 80, 81, 246, 115, 177, 225, 159]

A.4. Example Key Derivation with Outputs >= Hash Size

TOC

This example uses the Concat KDF to derive the Content Encryption Key (CEK) and Content Integrity Key (CIK) from the Content Master Key (CMK) in the manner described in Section 4.12 of **[JWA]**. In this example, a 512 bit CMK is used to derive a 256 bit CEK and a 512 bit CIK.

The CMK value is:

[148, 116, 199, 126, 2, 117, 233, 76, 150, 149, 89, 193, 61, 34, 239, 226, 109, 71, 59, 160, 192, 140, 150, 235, 106, 204, 49, 176, 68, 119, 13, 34, 49, 19, 41, 69, 5, 20, 252, 145, 104, 129, 137, 138, 67, 23, 153, 83, 81, 234, 82, 247, 48, 211, 41, 130, 35, 124, 45, 156, 249, 7, 225, 168]

A.4.1. CEK Generation

TOC

When deriving the CEK from the CMK, the ASCII label "Encryption" ([69, 110, 99, 114, 121, 112, 116, 105, 111, 110]) is used. The input to the first hash round is the concatenation of the big endian number 1 ([0, 0, 0, 1]), the CMK, and the label. Thus the round 1 hash input is:

[0, 0, 0, 1, 148, 116, 199, 126, 2, 117, 233, 76, 150, 149, 89, 193, 61, 34, 239, 226, 109, 71, 59, 160, 192, 140, 150, 235, 106, 204, 49, 176, 68, 119, 13, 34, 49, 19, 41, 69, 5, 20, 252, 145, 104, 129, 137, 138, 67, 23, 153, 83, 81, 234, 82, 247, 48, 211, 41, 130, 35, 124, 45, 156, 249, 7, 225, 168, 69, 110, 99, 114, 121, 112, 116, 105, 111, 110]

The SHA-256 hash of this value, which is the round 1 hash output, is:

[137, 5, 92, 9, 17, 47, 17, 86, 253, 235, 34, 247, 121, 78, 11, 144, 10, 172, 38, 247, 108, 243, 201, 237, 95, 80, 49, 150, 116, 240, 159, 64]

Given that 256 bits are needed for the CEK and the hash has produced 256 bits, the CEK value is that same value:

[137, 5, 92, 9, 17, 47, 17, 86, 253, 235, 34, 247, 121, 78, 11, 144, 10, 172, 38, 247, 108, 243, 201, 237, 95, 80, 49, 150, 116, 240, 159, 64]

A.4.2. CIK Generation

TOC

When deriving the CIK from the CMK, the ASCII label "Integrity" ([73, 110, 116, 101, 103, 114, 105, 116, 121]) is used. The input to the first hash round is the concatenation of the big endian number 1 ([0, 0, 0, 1]), the CMK, and the label. Thus the round 1 hash input is:

[0, 0, 0, 1, 148, 116, 199, 126, 2, 117, 233, 76, 150, 149, 89, 193, 61, 34, 239, 226, 109, 71, 59, 160, 192, 140, 150, 235, 106, 204, 49, 176, 68, 119, 13, 34, 49, 19, 41, 69, 5, 20, 252, 145, 104, 129, 137, 138, 67, 23, 153, 83, 81, 234, 82, 247, 48, 211, 41, 130, 35, 124, 45, 156, 249, 7, 225, 168, 73, 110, 116, 101, 103, 114, 105, 116, 121]

The SHA-256 hash of this value, which is the round 1 hash output, is:

[11, 179, 132, 177, 171, 24, 126, 19, 113, 1, 200, 102, 100, 74, 88, 149, 31, 41, 71, 57, 51, 179, 106, 242, 113, 211, 56, 56, 37, 198, 57, 17]

Given that 512 bits are needed for the CIK and the hash has produced only 256 bits, another round is needed. The input to the second hash round is the concatenation of the big endian number 2 ([0, 0, 0, 2]), the CMK, and the label. Thus the round 2 hash input is:

[0, 0, 0, 2, 148, 116, 199, 126, 2, 117, 233, 76, 150, 149, 89, 193, 61, 34, 239, 226, 109, 71, 59, 160, 192, 140, 150, 235, 106, 204, 49, 176, 68, 119, 13, 34, 49, 19, 41, 69, 5, 20, 252, 145, 104, 129, 137, 138, 67, 23, 153, 83, 81, 234, 82, 247, 48, 211, 41, 130, 35, 124, 45, 156, 249, 7, 225, 168, 73, 110, 116, 101, 103, 114, 105, 116, 121]

The SHA-256 hash of this value, which is the round 2 hash output, is:

[149, 209, 221, 113, 40, 191, 95, 252, 142, 254, 141, 230, 39, 113, 139, 84, 44, 156, 247, 47, 223, 101, 229, 180, 82, 231, 38, 96, 170, 119, 236, 81]

Given that 512 bits are needed for the CIK and the two rounds have collectively produced 512 bits of output, the CIK is the concatenation of the round 1 and round 2 hash outputs, which is:

[11, 179, 132, 177, 171, 24, 126, 19, 113, 1, 200, 102, 100, 74, 88, 149, 31, 41, 71, 57, 51, 179, 106, 242, 113, 211, 56, 56, 37, 198, 57, 17, 149, 209, 221, 113, 40, 191, 95, 252, 142, 254, 141, 230, 39, 113, 139, 84, 44, 156, 247, 47, 223, 101, 229, 180, 82, 231, 38, 96, 170, 119, 236, 81]

Appendix B. Acknowledgements

TOC

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.CR-xmlenc-core1-20120313] and **RFC 5652** [RFC5652] as possible, while utilizing simple compact JSON-based data structures.

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

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

Appendix C. Document History

TOC

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

-04

- Refer to the registries as the primary sources of defined values and then secondarily reference the sections defining the initial contents of the registries.
- Normatively reference **XML Encryption 1.1** [W3C.CR-xmlenc-core1-20120313] for its security considerations.
- Reference draft-jones-jose-jwe-json-serialization instead of draft-jones-json-web-encryption-json-serialization.
- Described additional open issues.
- Applied editorial suggestions.

-03

- 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.
- Reordered encryption steps so that the Encoded JWE Header is always created before it is needed as an input to the AEAD "additional authenticated data" parameter.
- 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.
- Moved description of how to determine whether a header is for a JWS or a JWE from the JWT spec to the JWE spec.
- Added complete encryption examples for both AEAD and non-AEAD algorithms.
- Added complete key derivation examples.
- Added "Collision Resistant Namespace" to the terminology section.
- Reference ITU.X690.1994 for DER encoding.
- Added Registry Contents sections to populate registry values.
- Numerous editorial improvements.

-02

- When using AEAD 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 Integrity Value.
- Defined KDF output key sizes.
- Generalized text to allow key agreement to be employed as an alternative to key wrapping or key encryption.
- Changed compression algorithm from gzip to DEFLATE.
- Clarified that it is an error when a `kid` value is included and no matching key is found.
- Clarified that JWEs with duplicate Header Parameter Names MUST be rejected.
- Clarified the relationship between `typ` header parameter values and MIME types.
- Registered application/jwe MIME type and "JWE" typ header parameter value.
- 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).
- Added suggestion on defining additional header parameters such as `x5t#S256` in the future for certificate thumbprints using hash algorithms other than SHA-1.
- Specify RFC 2818 server identity validation, rather than RFC 6125 (paralleling the same decision in the OAuth specs).
- 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.
- Reformatted to give each header parameter its own section heading.

-01

- Added an integrity check for non-AEAD algorithms.
- Added `jpk` and `x5c` header parameters for including JWK public keys and X.509 certificate chains directly in the header.
- Clarified that this specification is defining the JWE Compact Serialization. Referenced the new JWE-JS spec, which defines the JWE JSON Serialization.
- Added text "New header parameters should be introduced sparingly since an implementation that does not understand a parameter MUST reject the JWE".
- 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.
- Made other editorial improvements suggested by JOSE working group participants.

-00

- Created the initial IETF draft based upon draft-jones-json-web-encryption-02 with no normative changes.
- Changed terminology to no longer call both digital signatures and HMACs "signatures".

Authors' Addresses

TOC

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: http://self-issued.info/

Eric Rescorla
RTFM, Inc.
Email: ekr@rtfm.com

Joe Hildebrand
Cisco Systems, Inc.
Email: jhildebr@cisco.com