JSON Web Encryption (JWE)  
draft-ietf-jose-json-web-encryption-02

Abstract

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

Requirements Language

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 RFC 2119 [RFC2119].

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 November 13, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
2. Terminology
3. JSON Web Encryption (JWE) Overview
   3.1. Example JWE with an Integrated Integrity Check
1. Introduction

JSON Web Encryption (JWE) is a compact encryption format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. It provides a wrapper for encrypted content using JSON [RFC4627] data structures. The JWE encryption mechanisms are independent of the type of content being encrypted. Cryptographic algorithms and identifiers used with this specification are enumerated 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.

2. Terminology

JSON Web Encryption (JWE)

A data structure representing an encrypted version of a Plaintext. 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
The encrypted version 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.

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 Names
The names of the members within the JWE Header.

Header Parameter Values
The values of the members within the 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.

Base64url Encoding
For the purposes of this specification, this term always refers to the URL- and filename-safe Base64 encoding described in [RFC4648], Section 5, with the (non URL-safe) ‘=’ padding characters omitted, as permitted by Section 3.2. (See Appendix B of [JWS] for notes on implementing base64url encoding without padding.)

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 as defined in [RFC3986].

3. JSON Web Encryption (JWE) Overview

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 contents 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

The following example JWE Header declares that:

- the Content Master Key is encrypted to the recipient using the RSA-PKCS1_1.5 algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES-256-GCM algorithm to produce the JWE Ciphertext,
- the specified 96 bit Initialization Vector with the base64url encoding __79_Pv6-fj39vX0 was used, and
- a JSON Web Key Set (JWK Set) representation of the public key used to encrypt the JWE is located at https://example.com/public_key.jwk.

```json
{"alg":"RSA1_5","enc":"A256GCM","iv":"__79_Pv6-fj39vX0","jku":"https://example.com/public_key.jwk"}
```

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):

```text
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXzc5X1B2Ni1mZyIsDQogImprdSI6Imh0dHBzOi8vZXhhbXBsZS55jb20vcHVibGljX2tl
```

TBD: Finish this example by showing generation of a Content Master Key (CMK), saying that the CMK is used as the CEK and there is no separate integrity check since AES GCM is an AEAD algorithm, using the recipient's key to encrypt the CMK to produce the JWE Encrypted Key, using the CEK, IV, Encoded JWE Header, and Encoded JWE Encrypted Key to encrypt the Plaintext to produce the Ciphertext and "authentication tag" value, base64url encoding these values, and assembling the result.

Concatenating these parts in the order Header.EncryptedKey.Ciphertext.IntegrityValue with period characters between the parts yields this complete JWE representation (with line breaks for display purposes only):

```text
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXzc5X1B2Ni1mZyIsDQogImprdSI6Imh0dHBzOi8vZXhhbXBsZS55jb20vcHVibGljX2tl
```

### 3.2. Example JWE with a Separate Integrity Check
The following example JWE Header declares that:

- the Content Master Key is encrypted to the recipient using the RSA-PKCS1_1.5 algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES-256-CBC algorithm to produce the JWE 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,
- the specified 128 bit Initialization Vector with the base64url encoding `AxY8DctDaGlshbGljb3RoZQ` was used, and
- the thumbprint of the X.509 certificate that corresponds to the key used to encrypt the JWE has the base64url encoding `7no0Pq-hJ1_hCvWh6IeYI2w9Q0`.

```json
{"alg":"RSA1_5",
"enc":"A256CBC",
"int":"HS256",
"iv": "AxY8DctDaGlshbGljb3RoZQ",
"x5t": "7no0Pq-hJ1_hCvWh6IeYI2w9Q0"}
```

Because AES CBC is not an AEAD algorithm (and so provides no integrated content integrity check), a separate integrity check value is used.

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):

```
eyJhbGciOiJSU0ExXzUiLCJhbGciOiJBRkIiLCJlbmMiOiJBMjU2Q0JDIiwNCiAiaW50IjoiSFMyNTYiLCJpdiI6Ik16LW1XaWNrb2xvQ2FmdG9wIiwNCiAieDV0IjoiN25vT1BxLWhKMV9oQ252V2g2SWVZSTJ30VEwIn0
```

TBD: Finish this example by showing generation of a Content Master Key (CMK), showing the derivation of the CEK and the CIK from the CMK, using the recipient's key to encrypt the CMK to produce the JWE Encrypted Key, using the CEK and IV to encrypt the Plaintext to produce the Ciphertext, showing the computation of the JWE Integrity Value, base64url encoding these values, and assembling the result.

```
eyJhbGciOiJSU0ExXzUiLCJhbGciOiJBRkIiLCJlbmMiOiJBMjU2Q0JDIiwNCiAiaW50IjoiSFMyNTYiLCJpdiI6Ik16LW1XaWNrb2xvQ2FmdG9wIiwNCiAieDV0IjoiN25vT1BxLWhKMV9oQ252V2g2SWVZSTJ30VEwIn0
```

TBD: encrypted_key_value_TBD

TBD: ciphertext_value_TBD

TBD: integrity_value_TBD

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 three classes of Header Parameter Names: Reserved Header Parameter Names, Public Header Parameter Names, and Private 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 [JWA]. 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 \texttt{alg} (algorithm) header parameter identifies the cryptographic algorithm used to secure the JWE Encrypted Key. A list of defined \texttt{alg} values for use with JWE is presented in Section 4.1 of the JSON Web Algorithms (JWA) [JWA] specification. The processing of the \texttt{alg} header parameter requires that the value MUST be one that is both supported and for which there exists a key for use with that algorithm associated with the intended recipient. The \texttt{alg} value is case sensitive. Its value MUST be a string containing a StringOrURI value. This header parameter is REQUIRED.

\texttt{alg} values SHOULD either be defined in the IANA JSON Web Signature and Encryption Algorithms registry [JWA] or be a URI that contains a collision resistant namespace.

4.1.2. "enc" (Encryption Method) Header Parameter

The \texttt{enc} (encryption method) header parameter identifies the symmetric encryption algorithm used to secure the Ciphertext. A list of defined \texttt{enc} values is presented in Section 4.2 of the JSON Web Algorithms (JWA) [JWA] specification. The processing of the \texttt{enc} (encryption method) header parameter requires that the value MUST be one that is supported. The \texttt{enc} value is case sensitive. Its value MUST be a string containing a StringOrURI value. This header parameter is REQUIRED.

\texttt{enc} values SHOULD either be defined in the IANA JSON Web Signature and Encryption Algorithms registry [JWA] or be a URI that contains a collision resistant namespace.

4.1.3. "int" (Integrity Algorithm) Header Parameter

The \texttt{int} (integrity algorithm) header parameter identifies the cryptographic algorithm used to safeguard the integrity of the Ciphertext and the parameters used to create it. A list of defined \texttt{int} values is presented in Section 4.3 of the JSON Web Algorithms (JWA) [JWA] specification. The \texttt{int} parameter uses the MAC subset of the algorithm values used by the JWS \texttt{alg} parameter. The \texttt{int} value is case sensitive. Its value MUST be a 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.

\texttt{int} values SHOULD either be defined in the IANA JSON Web Signature and Encryption Algorithms registry [JWA] or be a URI that contains a collision resistant namespace.

4.1.4. "iv" (Initialization Vector) Header Parameter

The \texttt{iv} (initialization vector) value for algorithms requiring it, represented as a base64url encoded string. This header parameter is OPTIONAL.

4.1.5. "epk" (Ephemeral Public Key) Header Parameter
The **epk** (ephemeral public key) value created by the originator for the use in ECDH-ES [RFC 6090](https://tools.ietf.org/html/rfc6090) encryption. This key is represented as a JSON Web Key [JWK] value, containing **crv** (curve), **x**, and **y** members. The inclusion of the **jwk** **alg** (algorithm) member is OPTIONAL. This header parameter is OPTIONAL.

### 4.1.6. "zip" (Compression Algorithm) Header Parameter

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.

### 4.1.7. "jku" (JWK Set URL) Header Parameter

The **jku** (JWK Set URL) header parameter is an absolute URL 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. The keys MUST be encoded as a JSON Web Key Set (JWK Set) as defined in the JSON Web Key (JWK) [JWK] specification. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS [RFC 2818] [RFC2818] [RFC 5246] [RFC5246]; the identity of the server MUST be validated, as per Section 3.1 of HTTP Over TLS [RFC2818]. This header parameter is OPTIONAL.

### 4.1.8. "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 key is represented as a JSON Web Key [JWK]. This header parameter is OPTIONAL.

### 4.1.9. "x5u" (X.509 URL) Header Parameter

The **x5u** (X.509 URL) header parameter is an absolute URL that refers to a resource for the X.509 public key certificate or certificate chain corresponding to the key used to encrypt 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 [RFC 1421] [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 [RFC 2818] [RFC2818] [RFC 5246] [RFC5246]; the identity of the server MUST be validated, as per Section 3.1 of HTTP Over TLS [RFC2818]. This header parameter is OPTIONAL.

### 4.1.10. "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 corresponding to the key used to encrypt 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 [JWA].
4.1.11. "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 corresponding to the key used to encrypt the JWE. The certificate or certificate chain is represented as an array of certificate values. Each value is a base64-encoded (not base64url encoded) DER/BER 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.

4.1.12. "kid" (Key ID) Header Parameter

The kid (key ID) header parameter is a hint indicating which key was used to encrypt the JWE. This 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 contents of the kid parameter is unspecified. Its value MUST be a string. This header parameter is OPTIONAL.

4.1.13. "typ" (Type) Header Parameter

The typ (type) header parameter is used to declare the type of the encrypted content. The type value JWE MAY be used to indicate that the encrypted content is a JWE. The typ value is case sensitive. Its value MUST be a string. This header parameter is OPTIONAL.

MIME Media Type RFC 2045 [RFC2045] values MAY be used as typ values.

typ values SHOULD either be defined in the IANA JSON Web Signature and Encryption "typ" Values registry [JWA] or be a URI that contains a collision resistant namespace.

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 defined in the IANA JSON Web Signature and Encryption Header Parameters registry [JWA] 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

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
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](https://tools.ietf.org/html/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 8) 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 (see Section 7) 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. Encrypt M using the CEK and IV to form the byte sequence C. If an AEAD algorithm is used, use 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.

8. Base64url encode C to create the Encoded JWE Ciphertext.

9. 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.

10. Base64url encode the bytes of the UTF-8 representation of the JWE Header to create the Encoded JWE Header.

11. If not using an AEAD algorithm, run the integrity algorithm (see Section 9) 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.

12. Base64url encode the JWE Integrity Value to create the Encoded JWE Integrity Value.

13. The four encoded parts, taken together, are the result. 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

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. Parse the four parts of the input (which are separated by period characters when using the JWE Compact Serialization) into the Encoded JWE Header, the Encoded JWE Encrypted Key, the Encoded JWE Ciphertext, and the Encoded JWE Integrity Value.

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](https://tools.ietf.org/html/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 (see Section 7) 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 concatenation of the Encoded JWE Header, a period (\(\cdot\)) 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 9) 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. Key Derivation

The key derivation process converts the CMK into a CEK and a CIK. It assumes as a primitive a Key Derivation Function (KDF) which notionally takes three arguments:

- **MasterKey:** The master key used to compute the individual use keys
- **Label:** The use key label, used to differentiate individual use keys
- **Length:** The desired length of the use key

The only KDF used in this document is the Concat KDF, as defined in Section 5.8.1 of [NIST.800-56A], where the Digest Method is SHA-256, the SuppPubInfo parameter is the Label, and the remaining OtherInfo parameters are the empty bit string.

To compute the CEK from the CMK, the ASCII label "Encryption" ([69, 110, 99, 114, 121, 112, 116, 105, 111, 110]) is used. Use the key size for the enc algorithm as the CEK desired key length.

To compute the CIK from the CMK, the ASCII label "Integrity" ([73, 110, 116, 101, 103, 114, 105, 116, 121]) is used. Use the minimum key size for the int algorithm (for instance, 256 bits for HS256) as the CIK desired key length.

8. CMK Encryption

JWE supports two forms of CMK encryption:

- Asymmetric encryption under the recipient's public key.
- Symmetric encryption under a shared key.

8.1. Asymmetric Encryption

In the asymmetric encryption mode, the CMK is encrypted under the recipient's public key. The asymmetric encryption modes defined for use with this in this specification are listed in Section 4.1 of the JSON Web Algorithms (JWA) [JWA] specification.

8.2. Symmetric Encryption

In the symmetric encryption mode, the CMK is encrypted under a symmetric key shared
between the sender and receiver. The symmetric encryption modes defined for use with this in this specification are listed in Section 4.1 of the JSON Web Algorithms (JWA) [JWA] specification.

9. Integrity Value Calculation

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 (normally HMAC SHA-256).
- The JWS Secured Input used is 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.

10. Encrypting JWEs with Cryptographic Algorithms

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 enumerates a set of cryptographic algorithms and identifiers to be used with this specification. Specifically, Section 4.1 enumerates a set of alg (algorithm) header parameter values, Section 4.2 enumerates a set of enc (encryption method) header parameter values, and Section 4.3 enumerates a set of int (integrity algorithm) 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.

11. IANA Considerations

11.1. Registration of application/jwe MIME Media Type

This specification registers the application/jwe MIME Media Type RFC 2045 [RFC2045].

Type name: application
Subtype name: jwe
Required parameters: jwe
Optional parameters: n/a
Encoding considerations: n/a
Security considerations: See the Security Considerations section of this document
Interoperability considerations: n/a
11.2. Registration of "JWE" Type Value

This specification registers the following typ header parameter value in the JSON Web Signature and Encryption "typ" Values registry established by the JSON Web Algorithms (JWA) [JWA] specification:

"typ" header parameter value: "JWE"

Abbreviation for MIME type: application/jwe

Change controller: IETF

Description: [[ this document ]]

12. Security Considerations

All the security considerations in the JWS specification also apply to this specification, other than those that are signature specific. 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.

13. Open Issues and Things To Be Done (TBD)

The following items remain to be done in this draft:

- Add examples, including a KDF and a key agreement example.

14. References

14.1. Normative References

14.2. Informative References


Appendix A. JWE Examples

This section provides several examples of JWEs.

A.1. JWE Example using TBD Algorithm

A.1.1. Encrypting

TBD: Demonstrate encryption steps with this algorithm

A.1.2. Decrypting

TBD: Demonstrate decryption steps with this algorithm

Appendix B. 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.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.

Appendix C. Document History

-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".

Appendix C. Document History

-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".
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