JOSE Working Group	M. Jones
Internet-Draft	Microsoft
Intended status: Standards Track	E. Rescorla
Expires: September 13, 2012	RTFM, Inc.
	J. Hildebrand
	Cisco Systems, Inc.
	March 12, 2012

JSON Web Encryption (JWE)
draft-ietf-jose-json-web-encryption-01

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 HMAC 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

Copyright Notice

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 provides a wrapper for encrypted content using JSON **RFC 4627** [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 HMAC capabilities are described in the separate JSON Web Signature (JWS) **[JWS]** specification.

2. Terminology

TOC

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.

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 an HMAC function to ensure the integrity of the Ciphertext and the parameters used to create it.

Content Master Key (CMK)
: A randomly generated key from which the CEK and CIK are derived, which is encrypted to the recipient as the JWE Encrypted Key.

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

JWE Encrypted Key

The Content Encryption Key (CEK) 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.

JWE Ciphertext
: A byte array containing the Ciphertext.

JWE Integrity Value
: A byte array containing a HMAC 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 **RFC 3629** [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 **RFC 4648** [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.)

TOC

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.

TOC

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 64-bit Initialization Vector with the base64url encoding `__79_Pv6-fg` was used, and
- a JSON Web Key (JWK) representation of the public key used to encrypt the JWE is located at `https://example.com/public_key.jwk`.

```
{"alg":"RSA1_5",
 "enc":"A256GCM",
 "iv":"__79_Pv6-fg",
 "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):

```
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXzc5
X1B2Ni1mZyIsDQogImprdSI6Imh0dHBzOi8vZXhhbXBsZS5jb20vcHVibGljX2tl
eS5qd2sifQ
```

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 CEK to encrypt the Plaintext to produce the Ciphertext, using the recipient's key to encrypt the CMK to produce the JWE Encrypted Key, 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):

```
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXzc5
X1B2Ni1mZyIsDQogImprdSI6Imh0dHBzOi8vZXhhbXBsZS5jb20vcHVibGljX2tl
eS5qd2sifQ
.
TBD_encrypted_key_value_TBD
.
TBD_ciphertext_value_TBD
.
```

TOC

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 64-bit Initialization Vector with the base64url encoding `Mz-mW_4JHfg` was used, and
- the thumbprint of the X.509 certificate that corresponds to the key used to encrypt the JWE has the base64url encoding `7noOPq-hJ1_hCnvWh6IeYI2w9Q0`.

```
{"alg":"RSA1_5",
 "enc":"A256CBC",
 "int":"HS256",
 "iv":"Mz-mW_4JHfg",
 "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"}
```

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):

```
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2Q0JDIiwNCiAiaW50IjoiSFMy
NTYiLA0KICJpdiI6Ik16LW1XXzRKSGZnIiwNCiAieDV0IjoiN25vT1BxLWhKMV9o
Q252V2g2SWVZSTJ3OVEwIn0
```

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

```
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2Q0JDIiwNCiAiaW50IjoiSFMy
NTYiLA0KICJpdiI6Ik16LW1XXzRKSGZnIiwNCiAieDV0IjoiN25vT1BxLWhKMV9o
Q252V2g2SWVZSTJ3OVEwIn0
.
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. Implementations MUST understand the entire contents of the header; otherwise, the JWE MUST be rejected.

4.1. Reserved Header Parameter Names

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

Header Parameter Name	JSON Value Type	Header Parameter Syntax	Header Parameter Semantics
alg	string	StringOrURI	The `alg` (algorithm) header parameter identifies the cryptographic algorithm used to secure the JWE Encrypted Key. A list of defined encryption `alg` values is presented in Section 4, Table 2 of the JSON Web Algorithms (JWA) **[JWA]** specification. The processing of the `alg` (algorithm) 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 `alg` value is case sensitive. This header parameter is REQUIRED.
enc	string	StringOrURI	The `enc` (encryption method) header parameter identifies the symmetric encryption algorithm used to secure the Ciphertext. A list of defined `enc` values is presented in Section 4, Table 3 of the JSON Web Algorithms (JWA) **[JWA]** specification. The processing of the `enc` (encryption method) header parameter requires that the value MUST be one that is supported. The `enc` value is case sensitive. This header parameter is REQUIRED.
			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 same values as the JWS `alg` parameter; a

int	string	StringOrURI	list of defined JWS `alg` values is presented in Section 3, Table 1 of the JSON Web Algorithms (JWA) **[JWA]** specification. 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.
iv	string	String	Initialization Vector (`iv`) value for algorithms requiring it, represented as a base64url encoded string. This header parameter is OPTIONAL.
epk	object	JWK Key Object	Ephemeral Public Key (`epk`) value created by the originator for the use in ECDH-ES **RFC 6090** [RFC6090] encryption. This key is represented in the same manner as a JSON Web Key **[JWK]** JWK Key Object value, containing `crv` (curve), `x`, and `y` members. The inclusion of the JWK Key Object `alg` (algorithm) member is OPTIONAL. This header parameter is OPTIONAL.
zip	string	String	Compression algorithm (`zip`) applied to the Plaintext before encryption, if any. This specification defines the value `GZIP` to refer to the encoding format produced by the file compression program "gzip" (GNU zip) as described in **[RFC1952]**; this format is a Lempel-Ziv coding (LZ77) with a 32 bit CRC. If no `zip` parameter is present, or its value is `none`, no compression is applied to the Plaintext before encryption. The `zip` value is case sensitive. This header parameter is OPTIONAL.
jku	string	URL	The `jku` (JSON Web Key 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 that was used to encrypt the JWE. The keys MUST be encoded as described 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] with server authentication **RFC 6125** [RFC6125]. This header parameter is OPTIONAL.
kid	string	String	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. The interpretation of the contents of the `kid` parameter is unspecified. This header parameter is OPTIONAL.
jpk	object	JWK Key Object	The `jpk` (JSON Public Key) header parameter is a public key that corresponds to the key that was used to encrypt the JWE. This key is represented in the same manner as a JSON Web Key **[JWK]** JWK Key Object value. This header parameter is OPTIONAL.
x5u	string	URL	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 encrypting 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] with server authentication **RFC 6125** [RFC6125]. This header parameter is OPTIONAL.
x5t	string	String	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 that corresponds to the key that was used to encrypt the JWE. This header parameter is OPTIONAL.
			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

x5c	array	ArrayOfStrings	is a base64-encoded (not base64url encoded) DER/BER PKIX certificate value. The certificate containing the public key of the entity encrypting 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.
typ	string	String	The `typ` (type) header parameter is used to declare the type of the encrypted content. The `typ` value is case sensitive. This header parameter is OPTIONAL.

Table 1: Reserved Header Parameter Definitions

Additional reserved header parameter names MAY be defined via the IANA JSON Web Encryption Header Parameters registry, as per **Section 11**. The syntax values used above are defined as follows:

Syntax Name	Syntax Definition
String	Any string value MAY be used.
StringOrURI	Any string value MAY be used but a value containing a ":" character MUST be a URI as defined in **RFC 3986** [RFC3986].
URL	A URL as defined in **RFC 1738** [RFC1738].
ArrayOfStrings	An array of string values.

Table 2: Header Parameter Syntax Definitions

TOC

4.2. Public Header Parameter Names

Additional header parameter names can be defined by those using JWE. However, in order to prevent collisions, any new header parameter name or algorithm value SHOULD either be defined in the IANA JSON Web Encryption Header Parameters registry or be defined as 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 since an implementation that does not understand a parameter MUST reject the JWE.

TOC

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.

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

TOC

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. Generate a random Content Master Key (CMK). The CMK MUST have a length at

least equal to that of the larger of the required encryption or integrity keys and MUST be generated randomly. See **RFC 4086** [RFC4086] for considerations on generating random values.
2. Encrypt the CMK for the recipient (see **Section 8**) and let the result be the JWE Encrypted Key.
3. Base64url encode the JWE Encrypted Key to create the Encoded JWE Encrypted Key.
4. Generate a random Initialization Vector (IV) (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. Serialize the (compressed) Plaintext into a bitstring M.
8. Encrypt M using the CEK and IV to form the bitstring C.
9. Base64url encode C to create the Encoded JWE Ciphertext.
10. 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.
11. Base64url encode the bytes of the UTF-8 representation of the JWE Header to create the Encoded JWE Header.
12. 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 empty byte string.
13. Base64url encode the JWE Integrity Value to create the Encoded JWE Integrity Value.
14. 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.

TOC

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** [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. Decrypt the JWE Encrypted Key to produce the Content Master Key (CMK).
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. 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 be empty.
9. Decrypt the binary representation of the JWE Ciphertext using the CEK.
10. Remove the Initialization Vector (IV) value from the decrypted result (if an IV was used).
11. Uncompress the result of the previous step, if a `zip` parameter was included.
12. Output the resulting Plaintext.

7. Key Derivation

TOC

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 length of the desired use key

The only KDF used in this document is the Concat KDF, as defined in **[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" is used.

To compute the CIK from the CMK, the ASCII label "Integrity" is used.

When AEAD algorithms are used the KDF element MUST NOT be present. When they are not used, it MUST be present.

8. CMK Encryption

TOC

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

TOC

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, Table 2 of the JSON Web Algorithms (JWA) **[JWA]** specification.

8.2. Symmetric Encryption

TOC

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, Table 2 of the JSON Web Algorithms (JWA) **[JWA]** specification. For GCM, the random 64-bit IV is prepended to the ciphertext.

9. 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 an HMAC 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 HMAC key.

The computed JWS Signature value is the resulting integrity value.

10. Encrypting JWEs with Cryptographic Algorithms

TOC

JWE uses cryptographic algorithms to encrypt the Content Encryption Key (CMK) and the Plaintext. The JSON Web Algorithms (JWA) **[JWA]** specification enumerates a set of cryptographic algorithms and identifiers to be used with this specification. Specifically, Section 4, Table 2 enumerates a set of `alg` (algorithm) header parameter values and Section 4, Table 3 enumerates a set of `enc` (encryption method) 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

TOC

This specification calls for:

- A new IANA registry entitled "JSON Web Encryption Header Parameters" for reserved header parameter names is defined in **Section 4.1**. Inclusion in the registry is RFC Required in the **RFC 5226** [RFC5226] sense for reserved JWE header parameter names that are intended to be interoperable between implementations. The registry will just record the reserved header parameter name and a pointer to the RFC that defines it. This specification defines inclusion of the header parameter names defined in **Table 1**.

12. Security Considerations

TOC

TBD: Lots of work to do here. We need to remember to look into any issues relating to security and JSON parsing. One wonders just how secure most JSON parsing libraries are. Were they ever hardened for security scenarios? If not, what kind of holes does that open up? Also, we need to walk through the JSON standard and see what kind of issues we have especially around comparison of names. For instance, comparisons of header parameter names and other parameters must occur after they are unescaped. Need to also put in text about: Importance of keeping secrets secret. Rotating keys. Strengths and weaknesses of the different algorithms.

TBD: Need to put in text about why strict JSON validation is necessary. Basically, that if malformed JSON is received then the intent of the sender is impossible to reliably discern. One example of malformed JSON that MUST be rejected is an object in which the same member name occurs multiple times.

TBD: We need a section on generating randomness in browsers - it's easy to screw up.

When utilizing TLS to retrieve information, the authority providing the resource MUST be authenticated and the information retrieved MUST be free from modification.

12.1. Unicode Comparison Security Issues

TOC

Header parameter names in JWEs are Unicode strings. For security reasons, the representations of these names must be compared verbatim after performing any escape processing (as per **RFC 4627** [RFC4627], Section 2.5).

This means, for instance, that these JSON strings must compare as being equal ("enc", "\u0065nc"), whereas these must all compare as being not equal to the first set or to each other ("ENC", "Enc", "en\u0043").

JSON strings MAY contain characters outside the Unicode Basic Multilingual Plane. For instance, the G clef character (U+1D11E) may be represented in a JSON string as "\uD834\uDD1E". Ideally, JWE implementations SHOULD ensure that characters outside the Basic Multilingual Plane are preserved and compared correctly; alternatively, if this is not possible due to these characters exercising limitations present in the underlying JSON implementation, then input containing them MUST be rejected.

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

TOC

The following items remain to be done in this draft:

- EDITORIAL: Give each header parameter definition its own section. This will let them appear in the index, will give space for examples when needed, and will get rid of the way-too-cramped tables.
- Consider adding the DEFLATE compression algorithm (which omits the ZLIB header and checksum fields) and so produces smaller results than `GZIP`.
- Provide a more robust description of the use of the Initialization Vector (IV), including listing which algorithms require an IV. (This list may belong in the JWA spec.) The current statement "For GCM, the random 64-bit IV is prepended to the ciphertext" in the Symmetric Encryption section is almost certainly out of place and insufficiently general.
- Finish the Security Considerations section.
- Consider which of the open issues from the JWS and JWT specs also apply here.
- Should the JWE Encrypted Key be moved to the header (which would add about 20 bytes to every JWE) or left in a separate period-separated segment to prevent double base64 encoding?

14. References

TOC

14.1. Normative References

TOC

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

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

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

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

[NIST-800-56A] National Institute of Standards and Technology (NIST), "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)," NIST PUB 800-56A, March 2007.

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

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738, December 1994 (TXT).

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3," RFC 1952, May 1996 (TXT, PS, PDF).

[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).

[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).

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 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**).

[RFC6090] McGrew, D., Igoe, K., and M. Salter, “**Fundamental Elliptic Curve Cryptography Algorithms**,” RFC 6090, February 2011 (**TXT**).

[RFC6125] Saint-Andre, P. and J. Hodges, “**Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)**,” RFC 6125, March 2011 (**TXT**).

14.2. 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)**,” March 2012.

[RFC5652] Housley, R., “**Cryptographic Message Syntax (CMS)**,” STD 70, RFC 5652, September 2009 (**TXT**).

[W3C.CR-xmlenc-core1-20110303] Hirsch, F., Roessler, T., Reagle, J., and D. Eastlake, “**XML Encryption Syntax and Processing Version 1.1**,” World Wide Web Consortium CR CR-xmlenc-core1-20110303, March 2011 (**HTML**).

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-core1-20110303] 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

-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