JOSE Working Group M. Jones
Internet-Draft Microsoft
Intended status: Standards Track E. Rescorla
Expires: July 19, 2012 RTFM, Inc.
J. Hildebrand
Cisco Systems, Inc.
January 16, 2012

JSON Web Encryption (JWE)
draft-ietf-jose-json-web-encryption-00

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].

Table of Contents

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 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

JSON Web Encryption (JWE) A data structure representing an encrypted version of a Plaintext. The structure consists of three parts: the JWE Header, the JWE Encrypted Key, and the JWE Ciphertext.

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 generated to encrypt the Plaintext for the recipient to produce the Ciphertext, 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.

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.

Header Parameter Names The names of the members within the JWE Header.

Header Parameter Values The values of the members within the JWE Header.

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.)

3. JSON Web Encryption (JWE) Overview

JWE represents encrypted content using JSON data structures and base64url encoding. The representation consists of three parts: the JWE Header, the JWE Encrypted Key, and the JWE Ciphertext. The three parts are base64url-encoded for transmission, and typically represented as the concatenation of the encoded strings in that order, with the three strings being separated by period ('.') characters.

JWE utilizes encryption to ensure the confidentiality of the contents of the Plaintext. JWE does not add a content integrity check if not provided by the underlying encryption algorithm. If such a check is needed, an algorithm providing it such as AES-GCM [NIST-800-38D] can be used, or alternatively, it can be provided through composition by encrypting a representation of the digitally signed or HMACed content.

3.1. Example JWE

The following example JWE Header declares that:

- o the Content Encryption Key is encrypted to the recipient using the RSA-PKCS1_1.5 algorithm to produce the JWE Encrypted Key,
- o the Plaintext is encrypted using the AES-256-GCM algorithm to produce the JWE Ciphertext,
- o the specified 64-bit Initialization Vector with the base64url encoding "__79_Pv6-fg" was used, and

o 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":"A256GCM",
 "iv":"__79_Pv6-fg",
 "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"}
```


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
X1B2Ni1mZyIsDQogIng1dCI6Ijdub09QcS1oSjFfaENudldoNkllWUkydzlRMCJ9

TBD: Finish this example by showing generation of a Content Encryption Key (CEK), using the CEK to encrypt the Plaintext to produce the Ciphertext (and base64url encoding it), and using the recipient's key to encrypt the CEK to produce the JWE Encrypted Key (and base64url encoding it).

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.

TBD: Describe the relationship between the JWS and JWE header parameters - especially the "alg" parameter, which can contain digital signature or HMAC algorithms (from JWS) or encryption algorithms (from JWE), and the key reference parameters "jku", "kid", "x5u", and "x5t".

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.
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.

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 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.
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 10. 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].

Table 2: Header Parameter Syntax Definitions

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, 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.

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

5. Message Encryption

The message encryption process is as follows:

1. Generate a random Content Encryption Key (CEK). The CEK MUST have a length at least equal to that of the required encryption keys and MUST be generated randomly. See RFC 4086 [RFC4086] for considerations on generating random values.

2. Encrypt the CEK for the recipient (see Section 7).

3. Generate a random IV (if required for the algorithm).

4. Compress the Plaintext if a "zip" parameter was included.

5. Serialize the (compressed) Plaintext into a bitstring M.

6. Encrypt M using the CEK and IV to form the bitstring C.

7. Set the Encoded JWE Ciphertext equal to the base64url encoded representation of C.

8. Create a JWE Header containing the encryption parameters used. Note that white space is explicitly allowed in the representation and no canonicalization is performed before encoding.

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

10. The three encoded parts, taken together, are the result of the encryption.

6. Message Decryption

The message decryption process is the reverse of the encryption process. If any of these steps fails, the JWE MUST be rejected.

1. The Encoded JWE Header, the Encoded JWE Encrypted Key, and the Encoded JWE Ciphertext MUST be successfully base64url decoded following the restriction that no padding characters have been used.

2. The resulting JWE Header MUST be completely valid JSON syntax conforming to RFC 4627 [RFC4627].

3. The resulting JWE Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported.

4. Verify that the JWE Header appears to reference a key known to the recipient.

5. Decrypt the JWE Encrypted Key to produce the CEK.

6. Decrypt the binary representation of the JWE Ciphertext using the CEK.

7. Uncompress the result of the previous step, if a "zip" parameter was included.

8. Output the result.

7. CEK Encryption

JWE supports two forms of CEK encryption:

- Asymmetric encryption under the recipient's public key.

- Symmetric encryption under a shared key.

7.1. Asymmetric Encryption

In the asymmetric encryption mode, the CEK 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.

7.2. Symmetric Encryption

In the symmetric encryption mode, the CEK 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.

8. Composition

This document does not specify a combination integrity and encrypted mode. However, because the contents of a message can be arbitrary, encryption and data origin authentication can be provided by recursively encapsulating multiple JWE and JWS messages. In general, senders SHOULD digitally sign or HMAC the message and then encrypt the result (thus encrypting the digital signature or HMAC). This prevents attacks in which the digital signature or HMAC is stripped, leaving just an encrypted message, as well as providing privacy for signers.

9. Encrypting JWEs with Cryptographic Algorithms

JWE uses cryptographic algorithms to encrypt the Content Encryption Key (CEK) 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.

10. IANA Considerations

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.

11. Security Considerations

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.

11.1. Unicode Comparison Security Issues

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.

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

The following items remain to be done in this draft:

- Describe the relationship between the JWE, JWS, and JWT header parameters. In particular, point out that the set of "alg" values defined by each must be compatible and non-overlapping.

- Consider whether we want to define composite integrity/encryption operations (as was the consensus to do at IIW, as documented at http://self-issued.info/?p=378). This would provide both confidentiality and integrity.

- Consider whether reusing the JWS "jku", "kid", "x5u", and "x5t" parameters is the right thing to do, particularly as it effectively precludes specifying composite operations.

- Consider whether to add parameters for directly including keys in the header, either as JWK Key Objects, or X.509 cert values, or both.

o Consider whether to add version numbers.

o Consider which of the open issues from the JWS and JWT specs also apply here.

o Think about how to best describe the concept currently described as "the bytes of the UTF-8 representation of". Possible terms to use instead of "bytes of" include "byte sequence", "octet series", and "octet sequence". Also consider whether we want to add an overall clarifying statement somewhere in each spec something like "every place we say 'the UTF-8 representation of X', we mean 'the bytes of the UTF-8 representation of X'". That would potentially allow us to omit the "the bytes of" part everywhere else.

o Finish the Security Considerations section.

o Write a note in the Security Considerations section about how "x5t" (x.509 certificate thumbprint) should be deprecated because of known problems with SHA-1.

o Should StringOrURI use IRIs rather than RFC 3986 URIs?

o Provide a more robust description of the use of the IV. 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.

13. References

13.1. Normative References

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

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

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

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

[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.

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

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

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

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[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.

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.

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

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

[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, <http://www.w3.org/TR/2011/CR-xmlenc-core1-20110303>.

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

-00

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

Authors' Addresses

Michael B. Jones
Microsoft

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

Eric Rescorla
RTFM, Inc.

Email: ekr@rtfm.com

Joe Hildebrand
Cisco Systems, Inc.

Email: jhildebr@cisco.com