

JOSE Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 13, 2012

M. Jones
Microsoft
May 12, 2012

**JSON Web Algorithms (JWA)
draft-ietf-jose-json-web-algorithms-02**

Abstract

The JSON Web Algorithms (JWA) specification enumerates cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS) and JSON Web Encryption (JWE) specifications.

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	4
2.	Terminology	4
3.	Cryptographic Algorithms for JWS	4
3.1.	"alg" (Algorithm) Header Parameter Values for JWS	4
3.2.	MAC with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512	5
3.3.	Digital Signature with RSA SHA-256, RSA SHA-384, or RSA SHA-512	6
3.4.	Digital Signature with ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512	7
3.5.	Creating a Plaintext JWS	9
3.6.	Additional Digital Signature/MAC Algorithms and Parameters	9
4.	Cryptographic Algorithms for JWE	9
4.1.	"alg" (Algorithm) Header Parameter Values for JWE	9
4.2.	"enc" (Encryption Method) Header Parameter Values for JWE	10
4.3.	"int" (Integrity Algorithm) Header Parameter Values for JWE	11
4.4.	Key Encryption with RSA using RSA-PKCS1-1.5 Padding	11
4.5.	Key Encryption with RSA using Optimal Asymmetric Encryption Padding (OAEP)	11
4.6.	Key Agreement with Elliptic Curve Diffie-Hellman Ephemeral Static (ECDH-ES)	12
4.7.	Key Encryption with AES Key Wrap	12
4.8.	Plaintext Encryption with AES Cipher Block Chaining (CBC) Mode	12
4.9.	Plaintext Encryption with AES Galois/Counter Mode (GCM)	12
4.10.	Integrity Calculation with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512	13
4.11.	Additional Encryption Algorithms and Parameters	13
5.	Cryptographic Algorithms for JWK	13
5.1.	"alg" (Algorithm Family) Parameter Values for JWK	14
5.2.	JWK Parameters for Elliptic Curve Keys	14
5.2.1.	"crv" (Curve) Parameter	14
5.2.2.	"x" (X Coordinate) Parameter	14
5.2.3.	"y" (Y Coordinate) Parameter	14
5.3.	JWK Parameters for RSA Keys	14
5.3.1.	"mod" (Modulus) Parameter	15
5.3.2.	"exp" (Exponent) Parameter	15
5.4.	Additional Key Algorithm Families and Parameters	15

Jones

Expires November 13, 2012

[Page 2]

- [6. IANA Considerations](#) [15](#)
- 6.1. JSON Web Signature and Encryption Header Parameters Registry [15](#)
- [6.2. JSON Web Signature and Encryption Algorithms Registry](#) [15](#)
- 6.3. JSON Web Signature and Encryption "typ" Values Registry . 16
- [6.4. JSON Web Key Parameters Registry](#) [16](#)
- [6.5. JSON Web Key Algorithm Families Registry](#) [16](#)
- [7. Security Considerations](#) [16](#)
- [8. Open Issues and Things To Be Done \(TBD\)](#) [17](#)
- [9. References](#) [17](#)
- [9.1. Normative References](#) [17](#)
- [9.2. Informative References](#) [18](#)
- [Appendix A. Digital Signature/MAC Algorithm Identifier Cross-Reference](#) [19](#)
- [Appendix B. Encryption Algorithm Identifier Cross-Reference](#) [21](#)
- [Appendix C. Acknowledgements](#) [25](#)
- [Appendix D. Document History](#) [25](#)
- Author's Address [26](#)

1. Introduction

The JSON Web Algorithms (JWA) specification enumerates cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] specifications. Enumerating the algorithms and identifiers for them in this specification, rather than in the JWS and JWE specifications, is intended to allow them to remain unchanged in the face of changes in the set of required, recommended, optional, and deprecated algorithms over time. This specification also describes the semantics and operations that are specific to these algorithms and algorithm families.

2. Terminology

This specification uses the terminology defined by the JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] specifications.

3. Cryptographic Algorithms for JWS

JWS uses cryptographic algorithms to digitally sign or MAC the contents of the JWS Header and the JWS Payload. The use of the following algorithms for producing JWSs is defined in this section.

3.1. "alg" (Algorithm) Header Parameter Values for JWS

The table below is the set of "alg" (algorithm) header parameter values defined by this specification for use with JWS, each of which is explained in more detail in the following sections:

alg Parameter Value	Digital Signature or MAC Algorithm
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm
RS256	RSA using SHA-256 hash algorithm
RS384	RSA using SHA-384 hash algorithm
RS512	RSA using SHA-512 hash algorithm
ES256	ECDSA using P-256 curve and SHA-256 hash algorithm
ES384	ECDSA using P-384 curve and SHA-384 hash algorithm

Jones

Expires November 13, 2012

[Page 4]

ES512	ECDSA using P-521 curve and SHA-512 hash	
	algorithm	
none	No digital signature or MAC value included	
+-----+	+-----+	+-----+

See [Appendix A](#) for a table cross-referencing the digital signature and MAC "alg" (algorithm) values used in this specification with the equivalent identifiers used by other standards and software packages.

Of these algorithms, only HMAC SHA-256 and "none" MUST be implemented by conforming JWS implementations. It is RECOMMENDED that implementations also support the RSA SHA-256 and ECDSA P-256 SHA-256 algorithms. Support for other algorithms and key sizes is OPTIONAL.

3.2. MAC with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512

Hash-based Message Authentication Codes (HMACs) enable one to use a secret plus a cryptographic hash function to generate a Message Authentication Code (MAC). This can be used to demonstrate that the MAC matches the hashed content, in this case the JWS Secured Input, which therefore demonstrates that whoever generated the MAC was in possession of the secret. The means of exchanging the shared key is outside the scope of this specification.

The algorithm for implementing and validating HMACs is provided in [RFC 2104](#) [[RFC2104](#)]. This section defines the use of the HMAC SHA-256, HMAC SHA-384, and HMAC SHA-512 cryptographic hash functions as defined in FIPS 180-3 [[FIPS.180-3](#)]. The "alg" (algorithm) header parameter values "HS256", "HS384", and "HS512" are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded HMAC value using the respective hash function.

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm.

The HMAC SHA-256 MAC is generated as follows:

1. Apply the HMAC SHA-256 algorithm to the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation) using the shared key to produce an HMAC value.
2. Base64url encode the resulting HMAC value.

The output is the Encoded JWS Signature for that JWS.

The HMAC SHA-256 MAC for a JWS is validated as follows:

1. Apply the HMAC SHA-256 algorithm to the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation) of the JWS using the shared key.
2. Base64url encode the resulting HMAC value.
3. If the Encoded JWS Signature and the base64url encoded HMAC value exactly match, then one has confirmation that the shared key was used to generate the HMAC on the JWS and that the contents of the JWS have not be tampered with.
4. If the validation fails, the JWS MUST be rejected.

Alternatively, the Encoded JWS Signature MAY be base64url decoded to produce the JWS Signature and this value can be compared with the computed HMAC value, as this comparison produces the same result as comparing the encoded values.

Securing content with the HMAC SHA-384 and HMAC SHA-512 algorithms is performed identically to the procedure for HMAC SHA-256 - just with correspondingly larger minimum key sizes and result values.

3.3. Digital Signature with RSA SHA-256, RSA SHA-384, or RSA SHA-512

This section defines the use of the RSASSA-PKCS1-v1_5 digital signature algorithm as defined in [RFC 3447 \[RFC3447\], Section 8.2](#) (commonly known as PKCS#1), using SHA-256, SHA-384, or SHA-512 as the hash function. The RSASSA-PKCS1-v1_5 algorithm is described in FIPS 186-3 [[FIPS.186-3](#)], Section 5.5, and the SHA-256, SHA-384, and SHA-512 cryptographic hash functions are defined in FIPS 180-3 [[FIPS.180-3](#)]. The "alg" (algorithm) header parameter values "RS256", "RS384", and "RS512" are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded RSA digital signature using the respective hash function.

A key of size 2048 bits or larger MUST be used with these algorithms.

Note that while [Section 8 of RFC 3447 \[RFC3447\]](#) explicitly calls for people not to adopt RSASSA-PKCS1 for new applications and instead requests that people transition to RSASSA-PSS, for interoperability reasons, this specification does use RSASSA-PKCS1 because it commonly implemented.

The RSA SHA-256 digital signature is generated as follows:

1. Generate a digital signature of the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation) using RSASSA-PKCS1-V1_5-SIGN and the SHA-

256 hash function with the desired private key. The output will be a byte array.

2. Base64url encode the resulting byte array.

The output is the Encoded JWS Signature for that JWS.

The RSA SHA-256 digital signature for a JWS is validated as follows:

1. Take the Encoded JWS Signature and base64url decode it into a byte array. If decoding fails, the JWS MUST be rejected.
2. Submit the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation) and the public key corresponding to the private key used by the signer to the RSASSA-PKCS1-V1_5-VERIFY algorithm using SHA-256 as the hash function.
3. If the validation fails, the JWS MUST be rejected.

Signing with the RSA SHA-384 and RSA SHA-512 algorithms is performed identically to the procedure for RSA SHA-256 - just with correspondingly larger result values.

3.4. Digital Signature with ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined by FIPS 186-3 [[FIPS.186-3](#)]. ECDSA provides for the use of Elliptic Curve cryptography, which is able to provide equivalent security to RSA cryptography but using shorter key sizes and with greater processing speed. This means that ECDSA digital signatures will be substantially smaller in terms of length than equivalently strong RSA digital signatures.

This specification defines the use of ECDSA with the P-256 curve and the SHA-256 cryptographic hash function, ECDSA with the P-384 curve and the SHA-384 hash function, and ECDSA with the P-521 curve and the SHA-512 hash function. The P-256, P-384, and P-521 curves are also defined in FIPS 186-3. The "alg" (algorithm) header parameter values "ES256", "ES384", and "ES512" are used in the JWS Header to indicate that the Encoded JWS Signature contains a base64url encoded ECDSA P-256 SHA-256, ECDSA P-384 SHA-384, or ECDSA P-521 SHA-512 digital signature, respectively.

A key of size 160 bits or larger MUST be used with these algorithms.

The ECDSA P-256 SHA-256 digital signature is generated as follows:

1. Generate a digital signature of the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation) using ECDSA P-256 SHA-256 with the desired private key. The output will be the EC point (R, S), where R and S are unsigned integers.
2. Turn R and S into byte arrays in big endian order. Each array will be 32 bytes long.
3. Concatenate the two byte arrays in the order R and then S.
4. Base64url encode the resulting 64 byte array.

The output is the Encoded JWS Signature for the JWS.

The ECDSA P-256 SHA-256 digital signature for a JWS is validated as follows:

1. Take the Encoded JWS Signature and base64url decode it into a byte array. If decoding fails, the JWS MUST be rejected.
2. The output of the base64url decoding MUST be a 64 byte array.
3. Split the 64 byte array into two 32 byte arrays. The first array will be R and the second S (with both being in big endian byte order).
4. Submit the bytes of the UTF-8 representation of the JWS Secured Input (which is the same as the ASCII representation), R, S and the public key (x, y) to the ECDSA P-256 SHA-256 validator.
5. If the validation fails, the JWS MUST be rejected.

The ECDSA validator will then determine if the digital signature is valid, given the inputs. Note that ECDSA digital signature contains a value referred to as K, which is a random number generated for each digital signature instance. This means that two ECDSA digital signatures using exactly the same input parameters will output different signature values because their K values will be different. The consequence of this is that one must validate an ECDSA digital signature by submitting the previously specified inputs to an ECDSA validator.

Signing with the ECDSA P-384 SHA-384 and ECDSA P-521 SHA-512 algorithms is performed identically to the procedure for ECDSA P-256 SHA-256 - just with correspondingly larger result values.

[3.5.](#) Creating a Plaintext JWS

To support use cases where the content is secured by a means other than a digital signature or MAC value, JWSs MAY also be created without them. These are called "Plaintext JWSs". Plaintext JWSs MUST use the "alg" value "none", and are formatted identically to other JWSs, but with an empty JWS Signature value.

[3.6.](#) Additional Digital Signature/MAC Algorithms and Parameters

Additional algorithms MAY be used to protect JWSs with corresponding "alg" (algorithm) header parameter values being defined to refer to them. New "alg" header parameter values SHOULD either be defined in the IANA JSON Web Signature and Encryption Algorithms registry [Section 6.2](#) or be a URI that contains a collision resistant namespace. In particular, it is permissible to use the algorithm identifiers defined in XML DSIG [[RFC3275](#)], XML DSIG 2.0 [[W3C.CR-xmldsig-core2-20120124](#)], and related specifications as "alg" values.

As indicated by the common registry, JWSs and JWEs share a common "alg" value space. The values used by the two specifications MUST be distinct, as the "alg" value MAY be used to determine whether the object is a JWS or JWE.

Likewise, additional reserved header parameter names MAY be defined via the IANA JSON Web Signature and Encryption Header Parameters registry [Section 6.1](#). 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.](#) Cryptographic Algorithms for JWE

JWE uses cryptographic algorithms to encrypt the Content Master Key (CMK) and the Plaintext. This section specifies a set of specific algorithms for these purposes.

[4.1.](#) "alg" (Algorithm) Header Parameter Values for JWE

The table below is the set of "alg" (algorithm) header parameter values that are defined by this specification for use with JWE. These algorithms are used to encrypt the CMK, producing the JWE Encrypted Key, or to use key agreement to agree upon the CMK.

alg	Key Encryption or Agreement Algorithm
Parameter	
Value	
RSA1_5	RSA using RSA-PKCS1-1.5 padding, as defined in RFC 3447 [RFC3447]
RSA-OAEP	RSA using Optimal Asymmetric Encryption Padding (OAEP), as defined in RFC 3447 [RFC3447]
ECDH-ES	Elliptic Curve Diffie-Hellman Ephemeral Static, as defined in RFC 6090 [RFC6090], and using the Concat KDF, as defined in Section 5.8.1 of [NIST.800-56A], where the Digest Method is SHA-256 and all OtherInfo parameters are the empty bit string
A128KW	Advanced Encryption Standard (AES) Key Wrap Algorithm using 128 bit keys, as defined in RFC 3394 [RFC3394]
A256KW	Advanced Encryption Standard (AES) Key Wrap Algorithm using 256 bit keys, as defined in RFC 3394 [RFC3394]

4.2. "enc" (Encryption Method) Header Parameter Values for JWE

The table below is the set of "enc" (encryption method) header parameter values that are defined by this specification for use with JWE. These algorithms are used to encrypt the Plaintext, which produces the Ciphertext.

enc	Block Encryption Algorithm
Parameter	
Value	
A128CBC	Advanced Encryption Standard (AES) using 128 bit keys in Cipher Block Chaining (CBC) mode using PKCS #5 padding, as defined in [FIPS.197] and [NIST.800-38A]
A256CBC	Advanced Encryption Standard (AES) using 256 bit keys in Cipher Block Chaining (CBC) mode using PKCS #5 padding, as defined in [FIPS.197] and [NIST.800-38A]
A128GCM	Advanced Encryption Standard (AES) using 128 bit keys in Galois/Counter Mode (GCM), as defined in [FIPS.197] and [NIST.800-38D]
A256GCM	Advanced Encryption Standard (AES) using 256 bit keys in Galois/Counter Mode (GCM), as defined in [FIPS.197] and [NIST.800-38D]

See [Appendix B](#) for a table cross-referencing the encryption "alg" (algorithm) and "enc" (encryption method) values used in this

Jones

Expires November 13, 2012

[Page 10]

specification with the equivalent identifiers used by other standards and software packages.

Of these "alg" and "enc" algorithms, only RSA-PKCS1-1.5 with 2048 bit keys, AES-128-KW, AES-256-KW, AES-128-CBC, and AES-256-CBC MUST be implemented by conforming JWE implementations. It is RECOMMENDED that implementations also support ECDH-ES with 256 bit keys, AES-128-GCM, and AES-256-GCM. Support for other algorithms and key sizes is OPTIONAL.

4.3. "int" (Integrity Algorithm) Header Parameter Values for JWE

The table below is the set of "int" (integrity algorithm) header parameter values defined by this specification for use with JWE. Note that these are the HMAC SHA subset of the "alg" (algorithm) header parameter values defined for use with JWS [Section 3.1](#). />

int Parameter Value	Algorithm
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm

Of these "int" algorithms, only HMAC SHA-256 MUST be implemented by conforming JWE implementations. It is RECOMMENDED that implementations also support the RSA SHA-256 and ECDSA P-256 SHA-256 algorithms.

4.4. Key Encryption with RSA using RSA-PKCS1-1.5 Padding

This section defines the specifics of encrypting a JWE CMK with RSA using RSA-PKCS1-1.5 padding, as defined in [RFC 3447](#) [[RFC3447](#)]. The "alg" header parameter value "RSA1_5" is used in this case.

A key of size 2048 bits or larger MUST be used with this algorithm.

4.5. Key Encryption with RSA using Optimal Asymmetric Encryption Padding (OAEP)

This section defines the specifics of encrypting a JWE CMK with RSA using Optimal Asymmetric Encryption Padding (OAEP), as defined in [RFC 3447](#) [[RFC3447](#)]. The "alg" header parameter value "RSA-OAEP" is used in this case.

A key of size 2048 bits or larger MUST be used with this algorithm.

4.6. Key Agreement with Elliptic Curve Diffie-Hellman Ephemeral Static (ECDH-ES)

This section defines the specifics of agreeing upon a JWE CMK with Elliptic Curve Diffie-Hellman Ephemeral Static, as defined in [RFC 6090 \[RFC6090\]](#), and using the Concat KDF, as defined in Section 5.8.1 of [\[NIST.800-56A\]](#), where the Digest Method is SHA-256 and all OtherInfo parameters are the empty bit string. The "alg" header parameter value "ECDH-ES" is used in this case.

A key of size 160 bits or larger MUST be used for the Elliptic Curve keys used with this algorithm. The output of the Concat KDF MUST be a key of the same length as that used by the "enc" algorithm.

An "epk" (ephemeral public key) value MUST only be used for a single key agreement transaction.

4.7. Key Encryption with AES Key Wrap

This section defines the specifics of encrypting a JWE CMK with the Advanced Encryption Standard (AES) Key Wrap Algorithm using 128 or 256 bit keys, as defined in [RFC 3394 \[RFC3394\]](#). The "alg" header parameter values "A128KW" or "A256KW" are used in this case.

4.8. Plaintext Encryption with AES Cipher Block Chaining (CBC) Mode

This section defines the specifics of encrypting the JWE Plaintext with Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode using PKCS #5 padding using 128 or 256 bit keys, as defined in [\[FIPS.197\]](#) and [\[NIST.800-38A\]](#). The "enc" header parameter values "A128CBC" or "A256CBC" are used in this case.

Use of an Initialization Vector (IV) of size 128 bits is RECOMMENDED with this algorithm.

4.9. Plaintext Encryption with AES Galois/Counter Mode (GCM)

This section defines the specifics of encrypting the JWE Plaintext with Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) using 128 or 256 bit keys, as defined in [\[FIPS.197\]](#) and [\[NIST.800-38D\]](#). The "enc" header parameter values "A128GCM" or "A256GCM" are used in this case.

Use of an Initialization Vector (IV) of size 96 bits is REQUIRED with this algorithm.

The "additional authenticated data" parameter value for the encryption is the concatenation of the Encoded JWE Header, a period

Jones

Expires November 13, 2012

[Page 12]

('.') character, and the Encoded JWE Encrypted Key.

The requested size of the "authentication tag" output MUST be the same as the key size (for instance, 128 bits for "A128GCM").

As GCM is an AEAD algorithm, the JWE Integrity Value is set to be the "authentication tag" value produced by the encryption.

4.10. Integrity Calculation with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512

This section defines the specifics of computing a JWE Integrity Value with HMAC SHA-256, HMAC SHA-384, or HMAC SHA-512 as defined in FIPS 180-3 [[FIPS.180-3](#)]. The "int" header parameter values "HS256", "HS384", or "HS512" are used in this case.

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm.

4.11. Additional Encryption Algorithms and Parameters

Additional algorithms MAY be used to protect JWEs with corresponding "alg" (algorithm), "enc" (encryption method), and "int" (integrity algorithm) header parameter values being defined to refer to them. New "alg", "enc", and "int" header parameter values SHOULD either be defined in the IANA JSON Web Signature and Encryption Algorithms registry [Section 6.2](#) or be a URI that contains a collision resistant namespace. In particular, it is permissible to use the algorithm identifiers defined in XML Encryption [[W3C.REC-xmlenc-core-20021210](#)], XML Encryption 1.1 [[W3C.CR-xmlenc-core1-20120313](#)], and related specifications as "alg", "enc", and "int" values.

As indicated by the common registry, JWSs and JWEs share a common "alg" value space. The values used by the two specifications MUST be distinct, as the "alg" value MAY be used to determine whether the object is a JWS or JWE.

Likewise, additional reserved header parameter names MAY be defined via the IANA JSON Web Signature and Encryption Header Parameters registry [Section 6.1](#). 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.

5. Cryptographic Algorithms for JWK

A JSON Web Key (JWK) [[JWK](#)] is a JSON data structure that represents a

public key. A JSON Web Key Set (JWK Set) is a JSON data structure for representing a set of JWKs. This section specifies a set of algorithm families to be used for those public keys and the algorithm family specific parameters for representing those keys.

5.1. "alg" (Algorithm Family) Parameter Values for JWK

The table below is the set of "alg" (algorithm family) parameter values that are defined by this specification for use in JWKs.

alg Parameter Value	Algorithm Family
EC	Elliptic Curve [FIPS.186-3] key family
RSA	RSA [RFC3447] key family

5.2. JWK Parameters for Elliptic Curve Keys

JWKs can represent Elliptic Curve [FIPS.186-3] keys. In this case, the "alg" member value MUST be "EC". Furthermore, these additional members MUST be present:

5.2.1. "crv" (Curve) Parameter

The "crv" (curve) member identifies the cryptographic curve used with the key. Values defined by this specification are "P-256", "P-384" and "P-521". Additional "crv" values MAY be used, provided they are understood by implementations using that Elliptic Curve key. The "crv" value is case sensitive. Its value MUST be a string.

5.2.2. "x" (X Coordinate) Parameter

The "x" (x coordinate) member contains the x coordinate for the elliptic curve point. It is represented as the base64url encoding of the coordinate's big endian representation.

5.2.3. "y" (Y Coordinate) Parameter

The "y" (y coordinate) member contains the y coordinate for the elliptic curve point. It is represented as the base64url encoding of the coordinate's big endian representation.

5.3. JWK Parameters for RSA Keys

JWKs can represent RSA [RFC3447] keys. In this case, the "alg" member value MUST be "RSA". Furthermore, these additional members MUST be present:

[5.3.1.](#) "mod" (Modulus) Parameter

The "mod" (modulus) member contains the modulus value for the RSA public key. It is represented as the base64url encoding of the value's big endian representation.

[5.3.2.](#) "exp" (Exponent) Parameter

The "exp" (exponent) member contains the exponent value for the RSA public key. It is represented as the base64url encoding of the value's big endian representation.

[5.4.](#) Additional Key Algorithm Families and Parameters

Public keys using additional algorithm families MAY be represented using JWK data structures with corresponding "alg" (algorithm family) parameter values being defined to refer to them. New "alg" parameter values SHOULD either be defined in the IANA JSON Web Key Algorithm Families registry [Section 6.5](#) or be a URI that contains a collision resistant namespace.

Likewise, parameters for representing keys for additional algorithm families or additional key properties SHOULD either be defined in the IANA JSON Web Key Parameters registry [Section 6.4](#) or be a URI that contains a collision resistant namespace.

[6.](#) IANA Considerations

[6.1.](#) JSON Web Signature and Encryption Header Parameters Registry

This specification establishes the IANA JSON Web Signature and Encryption Header Parameters registry for reserved JWS and JWE header parameter names. Inclusion in the registry is RFC Required in the [RFC 5226](#) [[RFC5226](#)] sense. The registry records the reserved header parameter name and a reference to the RFC that defines it. This specification registers the header parameter names defined in JSON Web Signature (JWS) [[JWS](#)], Section 4.1 and JSON Web Encryption (JWE) [[JWE](#)], Section 4.1.

[6.2.](#) JSON Web Signature and Encryption Algorithms Registry

This specification establishes the IANA JSON Web Signature and Encryption Algorithms registry for values of the JWS and JWE "alg" (algorithm), "enc" (encryption method), and "int" (integrity algorithm) header parameters. Inclusion in the registry is RFC Required in the [RFC 5226](#) [[RFC5226](#)] sense. The registry records the algorithm usage "alg", "enc", or "int", the value, and a pointer to

the RFC that defines it. This specification registers the values defined in [Section 3.1](#), [Section 4.1](#), [Section 4.2](#), and [Section 4.3](#).

6.3. JSON Web Signature and Encryption "typ" Values Registry

This specification establishes the IANA JSON Web Signature and Encryption "typ" Values registry for values of the JWS and JWE "typ" (type) header parameter. Inclusion in the registry is RFC Required in the [RFC 5226](#) [[RFC5226](#)] sense. It is RECOMMENDED that all registered "typ" values also register a MIME Media Type [RFC 2045](#) [[RFC2045](#)] that the registered value is a short name for. The registry records the "typ" value, the MIME type value that it is an abbreviation for (if any), and a pointer to the RFC that defines it.

MIME Media Type [RFC 2045](#) [[RFC2045](#)] values MUST NOT be directly registered as new "typ" values; rather, new "typ" values MAY be registered as short names for MIME types.

6.4. JSON Web Key Parameters Registry

This specification establishes the IANA JSON Web Key Parameters registry for reserved JWK parameter names. Inclusion in the registry is RFC Required in the [RFC 5226](#) [[RFC5226](#)] sense. The registry records the reserved parameter name and a reference to the RFC that defines it. This specification registers the parameter names defined in JSON Web Key (JWK) [[JWK](#)], [Section 4.2](#), JSON Web Encryption (JWE) [[JWE](#)], [Section 4.1](#), [Section 5.2](#), and [Section 5.3](#).

6.5. JSON Web Key Algorithm Families Registry

This specification establishes the IANA JSON Web Key Algorithm Families registry for values of the JWK "alg" (algorithm family) parameter. Inclusion in the registry is RFC Required in the [RFC 5226](#) [[RFC5226](#)] sense. The registry records the "alg" value and a pointer to the RFC that defines it. This specification registers the values defined in [Section 5.1](#).

7. Security Considerations

The security considerations in the JWS, JWE, and JWK specifications also apply to this specification.

Eventually the algorithms and/or key sizes currently described in this specification will no longer be considered sufficiently secure and will be removed. Therefore, implementers and deployments must be prepared for this eventuality.

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

The following items remain to be done in this draft:

- o Find values for encryption algorithm cross-reference table currently listed as "TBD" or determine that they do not exist.

9. References

9.1. Normative References

[FIPS.180-3]

National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-3, October 2008.

[FIPS.186-3]

National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.

[FIPS.197]

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

[JWE]

Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", May 2012.

[JWK]

Jones, M., "JSON Web Key (JWK)", May 2012.

[JWS]

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

[NIST.800-38A]

National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation", NIST PUB 800-38A, December 2001.

[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.

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", [RFC 2045](#), November 1996.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", [RFC 3394](#), September 2002.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", [RFC 3447](#), February 2003.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), May 2008.
- [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", [RFC 6090](#), February 2011.

9.2. Informative References

- [CanvasApp] Facebook, "Canvas Applications", 2010.
- [I-D.rescorla-jsms] Rescorla, E. and J. Hildebrand, "JavaScript Message Security Format", [draft-rescorla-jsms-00](#) (work in progress), March 2011.
- [JCA] Oracle, "Java Cryptography Architecture", 2011.
- [JSE] Bradley, J. and N. Sakimura (editor), "JSON Simple Encryption", September 2010.
- [JSS] Bradley, J. and N. Sakimura (editor), "JSON Simple Sign", September 2010.
- [MagicSignatures] Panzer (editor), J., Laurie, B., and D. Balfanz, "Magic Signatures", January 2011.

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", [RFC 3275](#), March 2002.

[W3C.CR-xmlsig-core2-20120124]
 Eastlake, D., Reagle, J., Yiu, K., Solo, D., Datta, P., Hirsch, F., Cantor, S., and T. Roessler, "XML Signature Syntax and Processing Version 2.0", World Wide Web Consortium CR CR-xmlsig-core2-20120124, January 2012, <<http://www.w3.org/TR/2012/CR-xmlsig-core2-20120124>>.

[W3C.CR-xmlenc-core1-20120313]
 Eastlake, D., Reagle, J., Roessler, T., and F. Hirsch, "XML Encryption Syntax and Processing Version 1.1", World Wide Web Consortium CR CR-xmlenc-core1-20120313, March 2012, <<http://www.w3.org/TR/2012/CR-xmlenc-core1-20120313>>.

[W3C.REC-xmlenc-core-20021210]
 Eastlake, D. and J. Reagle, "XML Encryption Syntax and Processing", World Wide Web Consortium Recommendation REC-xmlenc-core-20021210, December 2002, <<http://www.w3.org/TR/2002/REC-xmlenc-core-20021210>>.

Appendix A. Digital Signature/MAC Algorithm Identifier Cross-Reference

This appendix contains a table cross-referencing the digital signature and MAC "alg" (algorithm) values used in this specification with the equivalent identifiers used by other standards and software packages. See XML DSIG [[RFC3275](#)], XML DSIG 2.0 [[W3C.CR-xmlsig-core2-20120124](#)], and Java Cryptography Architecture [[JCA](#)] for more information about the names defined by those documents.

Algor ithm	JWS	XML DSIG	JCA	OID
HMAC using SHA-2 56 hash algo rithm	HS2 56	http://www.w3.org/2001/04/xmlsig-more#hmac-sha256	HmacSHA2 56	1.2.840.113 549.2.9

HMAC	HS3	http://www.w3.org/2001/04/	HmacSHA3	1.2.840.113
using	84	xmlsig-more#hmac-sha384	84	549.2.10
SHA-3				
84				
hash				
algo				
rithm				
HMAC	HS5	http://www.w3.org/2001/04/	HmacSHA5	1.2.840.113
using	12	xmlsig-more#hmac-sha512	12	549.2.11
SHA-5				
12				
hash				
algo				
rithm				
RSA	RS2	http://www.w3.org/2001/04/	SHA256wi	1.2.840.113
using	56	xmlsig-more#rsa-sha256	thRSA	549.1.1.11
SHA-2				
56				
hash				
algo				
rithm				
RSA	RS3	http://www.w3.org/2001/04/	SHA384wi	1.2.840.113
using	84	xmlsig-more#rsa-sha384	thRSA	549.1.1.12
SHA-3				
84				
hash				
algo				
rithm				
RSA	RS5	http://www.w3.org/2001/04/	SHA512wi	1.2.840.113
using	12	xmlsig-more#rsa-sha512	thRSA	549.1.1.13
SHA-5				
12				
hash				
algo				
rithm				
ECDSA	ES2	http://www.w3.org/2001/04/	SHA256wi	1.2.840.100
using	56	xmlsig-more#ecdsa-sha256	thECDSA	45.4.3.2
P-256				
curve				
and				
SHA-2				
56				
hash				
algo				
rithm				

ECDSA	ES3	http://www.w3.org/2001/04/	SHA384wi	1.2.840.100
using	84	xmldsig-more#ecdsa-sha384	thECDSA	45.4.3.3
P-384				
curve				
and				
SHA-3				
84				
hash				
algo				
rithm				
ECDSA	ES5	http://www.w3.org/2001/04/	SHA512wi	1.2.840.100
using	12	xmldsig-more#ecdsa-sha512	thECDSA	45.4.3.4
P-521				
curve				
and				
SHA-5				
12				
hash				
algo				
rithm				

Appendix B. Encryption Algorithm Identifier Cross-Reference

This appendix contains a table cross-referencing the "alg" (algorithm) and "enc" (encryption method) values used in this specification with the equivalent identifiers used by other standards and software packages. See XML Encryption [[W3C.REC-xmlenc-core-20021210](#)], XML Encryption 1.1 [[W3C.CR-xmlenc-core1-20120313](#)], and Java Cryptography Architecture [[JCA](#)] for more information about the names defined by those documents.

Algorithm	JWE	XML ENC	JCA
hmac			
RSA	RSA1_	http://www.w3.org/2001/04/	RSA/ECB/PKCS1Paddin
using	5	/xmlenc#rsa-1_5	g
RSA-PKC			
S1-1.5			
paddin			
g			

Advance d Encryp tion Stand ard(AES) Key Wrap Algo rithm R FC 339 4 [RF C3394] using25 6 bitke ys	A256K W	http://www.w3.org/2001/04/xmlenc#kw-aes256	TBD
Advance d Encryp tion Stand ard(AES) usin g 128 bitkeys inCiph er Bloc k Chai ning(CB C) mod e usi ng PKC S #5pad ding	A128C BC	http://www.w3.org/2001/04/xmlenc#aes128-cbc	AES/CBC/PKCS5Paddin g

Advance d Encryp tion Stand ard(AES) usin g 256 bitkeys inCiph er Bloc k Chai ning(CB C) mod e usi ng PKC S #5pad ding	A256C BC	http://www.w3.org/2001/04/xmlenc#aes256-cbc	AES/CBC/PKCS5Paddin g
Advance d Encryp tion Stand ard(AES) usin g 128 bitkeys inGalo is/Coun ter Mod e (GC M)	A128G CM	http://www.w3.org/2009/xmlenc11#aes128-gcm	AES/GCM/NoPadding
Advance d Encryp tion Stand ard(AES) usin g 256 bitkeys inGalo is/Coun ter Mod e (GC M)	A256G CM	http://www.w3.org/2009/xmlenc11#aes256-gcm	AES/GCM/NoPadding

[Appendix C](#). Acknowledgements

Solutions for signing and encrypting JSON content were previously explored by Magic Signatures [[MagicSignatures](#)], JSON Simple Sign [[JSS](#)], Canvas Applications [[CanvasApp](#)], JSON Simple Encryption [[JSE](#)], and JavaScript Message Security Format [[I-D.rescorla-jms](#)], all of which influenced this draft. Dirk Balfanz, John Bradley, Yaron Y. Golan, John Panzer, Nat Sakimura, and Paul Tarjan all made significant contributions to the design of this specification and its related specifications.

[Appendix D](#). Document History

-02

- o For 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.
- o Defined minimum required key sizes for algorithms without specified key sizes.
- o Defined KDF output key sizes.
- o Specified the use of PKCS #5 padding with AES-CBC.
- o Generalized text to allow key agreement to be employed as an alternative to key wrapping or key encryption.
- o Clarified that ECDH-ES is a key agreement algorithm.
- o Required implementation of AES-128-KW and AES-256-KW.
- o Removed the use of "A128GCM" and "A256GCM" for key wrapping.
- o Removed "A512KW" since it turns out that it's not a standard algorithm.
- o Clarified the relationship between "typ" header parameter values and MIME types.
- o Generalized language to refer to Message Authentication Codes (MACs) rather than Hash-based Message Authentication Codes (HMACs) unless in a context specific to HMAC algorithms.

- o Established registries: JSON Web Signature and Encryption Header Parameters, JSON Web Signature and Encryption Algorithms, JSON Web Signature and Encryption "typ" Values, JSON Web Key Parameters, and JSON Web Key Algorithm Families.
- o Moved algorithm-specific definitions from JWK to JWA.
- o Reformatted to give each member definition its own section heading.

-01

- o Moved definition of "alg":"none" for JWSs here from the JWT specification since this functionality is likely to be useful in more contexts than just for JWTs.
- o Added Advanced Encryption Standard (AES) Key Wrap Algorithm using 512 bit keys ("A512KW").
- o Added text "Alternatively, the Encoded JWS Signature MAY be base64url decoded to produce the JWS Signature and this value can be compared with the computed HMAC value, as this comparison produces the same result as comparing the encoded values".
- o Corrected the Magic Signatures reference.
- o Made other editorial improvements suggested by JOSE working group participants.

-00

- o Created the initial IETF draft based upon [draft-jones-json-web-signature-04](#) and [draft-jones-json-web-encryption-02](#) with no normative changes.
- o Changed terminology to no longer call both digital signatures and HMACs "signatures".

Author's Address

Michael B. Jones
Microsoft

Email: mbj@microsoft.com

URI: <http://self-issued.info/>

