Internet Engineering Task Force (IETF)
Request for Comments: 7518
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft

May 2015

JSON Web Algorithms (JWA)

Abstract

This specification registers cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS), JSON Web Encryption (JWE), and JSON Web Key (JWK) specifications. It defines several IANA registries for these identifiers.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7518.

Table of Contents

1. Introduction

This specification registers cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS) [JWS], JSON Web Encryption (JWE) [JWE], and JSON Web Key (JWK) [JWK] specifications. It defines several IANA registries for these identifiers. All these specifications utilize JSON-based [RFC7159] data structures. This specification also describes the semantics and operations that are specific to these algorithms and key types.

Registering the algorithms and identifiers here, rather than in the JWS, JWE, and JWK 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 also allows changes to the JWS, JWE, and JWK specifications without changing this document.

Names defined by this specification are short because a core goal is for the resulting representations to be compact.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

The interpretation should only be applied when the terms appear in all capital letters.

BASE64URL(OCTETS) denotes the base64url encoding of OCTETS, per Section 2 of [JWS].

UTF8(STRING) denotes the octets of the UTF-8 [RFC3629] representation of STRING, where STRING is a sequence of zero or more Unicode [UNICODE] characters.

ASCII(STRING) denotes the octets of the ASCII [RFC20] representation of STRING, where STRING is a sequence of zero or more ASCII characters.

The concatenation of two values A and B is denoted as A || B.

2. Terminology

The terms "JSON Web Signature (JWS)", "Base64url Encoding", "Header Parameter", "JOSE Header", "JWS Payload", "JWS Protected Header", "JWS Signature", "JWS Signing Input", and "Unsecured JWS" are defined by the JWS specification [JWS].

The terms "JSON Web Encryption (JWE)", "Additional Authenticated Data (AAD)", "Authentication Tag", "Content Encryption Key (CEK)", "Direct Encryption", "Direct Key Agreement", "JWE Authentication Tag", "JWE Ciphertext", "JWE Encrypted Key", "JWE Initialization Vector", "JWE Protected Header", "Key Agreement with Key Wrapping", "Key Encryption", "Key Management Mode", and "Key Wrapping" are defined by the JWE specification [JWE].

The terms "JSON Web Key (JWK)" and "JWK Set" are defined by the JWK specification [JWK].

The terms "Ciphertext", "Digital Signature", "Initialization Vector", "Message Authentication Code (MAC)", and "Plaintext" are defined by the "Internet Security Glossary, Version 2" [RFC4949].

This term is defined by this specification:

Base64urlUInt
: The representation of a positive or zero integer value as the base64url encoding of the value's unsigned big-endian representation as an octet sequence. The octet sequence MUST utilize the minimum number of octets needed to represent the value. Zero is represented as BASE64URL(single zero-valued octet), which is "AA".

3. Cryptographic Algorithms for Digital Signatures and MACs

JWS uses cryptographic algorithms to digitally sign or create a MAC of the contents of the JWS Protected Header and the JWS Payload.

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" Param Value	Digital Signature or MAC Algorithm	Implementation Requirements
HS256	HMAC using SHA-256	Required
HS384	HMAC using SHA-384	Optional
HS512	HMAC using SHA-512	Optional
RS256	RSASSA-PKCS1-v1_5 using SHA-256	Recommended
RS384	RSASSA-PKCS1-v1_5 using SHA-384	Optional
RS512	RSASSA-PKCS1-v1_5 using SHA-512	Optional
ES256	ECDSA using P-256 and SHA-256	Recommended+
ES384	ECDSA using P-384 and SHA-384	Optional
ES512	ECDSA using P-521 and SHA-512	Optional
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256	Optional
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384	Optional
PS512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512	Optional
none	No digital signature or MAC performed	Optional

The use of "+" in the Implementation Requirements column indicates that the requirement strength is likely to be increased in a future version of the specification.

See Appendix A.1 for a table cross-referencing the JWS digital signature and MAC "alg" (algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

3.2. HMAC with SHA-2 Functions

Hash-based Message Authentication Codes (HMACs) enable one to use a secret plus a cryptographic hash function to generate a MAC. This can be used to demonstrate that whoever generated the MAC was in possession of the MAC key. The algorithm for implementing and validating HMACs is provided in RFC 2104 [RFC2104].

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm. (This requirement is based on Section 5.3.4 (Security Effect of the HMAC Key) of NIST SP 800-117 [NIST.800-107], which states that the effective security strength is the minimum of the security strength of the key and two times the size of the internal hash value.)

The HMAC SHA-256 MAC is generated per RFC 2104, using SHA-256 as the hash algorithm "H", using the JWS Signing Input as the "text" value, and using the shared key. The HMAC output value is the JWS Signature.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWS Signature is an HMAC value computed using the corresponding algorithm:

"alg" Param Value	MAC Algorithm
HS256	HMAC using SHA-256
HS384	HMAC using SHA-384
HS512	HMAC using SHA-512

The HMAC SHA-256 MAC for a JWS is validated by computing an HMAC value per RFC 2104, using SHA-256 as the hash algorithm "H", using the received JWS Signing Input as the "text" value, and using the shared key. This computed HMAC value is then compared to the result of base64url decoding the received encoded JWS Signature value. The comparison of the computed HMAC value to the JWS Signature value MUST be done in a constant-time manner to thwart timing attacks. Alternatively, the computed HMAC value can be base64url encoded and compared to the received encoded JWS Signature value (also in a constant-time manner), as this comparison produces the same result as comparing the unencoded values. In either case, if the values match, the HMAC has been validated.

Securing content and validation with the HMAC SHA-384 and HMAC SHA-512 algorithms is performed identically to the procedure for HMAC SHA-256 -- just using the corresponding hash algorithms with correspondingly larger minimum key sizes and result values: 384 bits each for HMAC SHA-384 and 512 bits each for HMAC SHA-512.

An example using this algorithm is shown in Appendix A.1 of [JWS].

3.3. Digital Signature with RSASSA-PKCS1-v1_5

This section defines the use of the RSASSA-PKCS1-v1_5 digital signature algorithm as defined in Section 8.2 of RFC 3447 [RFC3447] (commonly known as PKCS #1), using SHA-2 [SHS] hash functions.

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

The RSASSA-PKCS1-v1_5 SHA-256 digital signature is generated as follows: generate a digital signature of the JWS Signing Input using RSASSA-PKCS1-v1_5-SIGN and the SHA-256 hash function with the desired private key. This is the JWS Signature value.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWS Signature is a digital signature value computed using the corresponding algorithm:

"alg" Param Value	Digital Signature Algorithm
RS256	RSASSA-PKCS1-v1_5 using SHA-256
RS384	RSASSA-PKCS1-v1_5 using SHA-384
RS512	RSASSA-PKCS1-v1_5 using SHA-512

The RSASSA-PKCS1-v1_5 SHA-256 digital signature for a JWS is validated as follows: submit the JWS Signing Input, the JWS Signature, 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.

Signing and validation with the RSASSA-PKCS1-v1_5 SHA-384 and RSASSA-PKCS1-v1_5 SHA-512 algorithms is performed identically to the procedure for RSASSA-PKCS1-v1_5 SHA-256 -- just using the corresponding hash algorithms instead of SHA-256.

An example using this algorithm is shown in Appendix A.2 of [JWS].

3.4. Digital Signature with ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) [DSS] 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 for many operations. 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 defined in [DSS].

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

1. Generate a digital signature of the JWS Signing Input using ECDSA P-256 SHA-256 with the desired private key. The output will be the pair (R, S), where R and S are 256-bit unsigned integers.

2. Turn R and S into octet sequences in big-endian order, with each array being be 32 octets long. The octet sequence representations MUST NOT be shortened to omit any leading zero octets contained in the values.

3. Concatenate the two octet sequences in the order R and then S. (Note that many ECDSA implementations will directly produce this concatenation as their output.)

4. The resulting 64-octet sequence is the JWS Signature value.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWS Signature is a digital signature value computed using the corresponding algorithm:

"alg" Param Value	Digital Signature Algorithm
ES256	ECDSA using P-256 and SHA-256
ES384	ECDSA using P-384 and SHA-384
ES512	ECDSA using P-521 and SHA-512

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

1. The JWS Signature value MUST be a 64-octet sequence. If it is not a 64-octet sequence, the validation has failed.

2. Split the 64-octet sequence into two 32-octet sequences. The first octet sequence represents R and the second S. The values R and S are represented as octet sequences using the Integer-to-OctetString Conversion defined in Section 2.3.7 of SEC1 [SEC1] (in big-endian octet order).

3. Submit the JWS Signing Input, R, S, and the public key (x, y) to the ECDSA P-256 SHA-256 validator.

Signing and validation 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 using the corresponding hash algorithms with correspondingly larger result values. For ECDSA P-384 SHA-384, R and S will be 384 bits each, resulting in a 96-octet sequence. For ECDSA P-521 SHA-512, R and S will be 521 bits each, resulting in a 132-octet sequence. (Note that the Integer-to-OctetString Conversion defined in Section 2.3.7 of SEC1 [SEC1] used to represent R and S as octet sequences adds zero-valued high-order padding bits when needed to round the size up to a multiple of 8 bits; thus, each 521-bit integer is represented using 528 bits in 66 octets.)

Examples using these algorithms are shown in Appendices A.3 and A.4 of [JWS].

3.5. Digital Signature with RSASSA-PSS

This section defines the use of the RSASSA-PSS digital signature algorithm as defined in Section 8.1 of RFC 3447 [RFC3447] with the MGF1 mask generation function and SHA-2 hash functions, always using the same hash function for both the RSASSA-PSS hash function and the MGF1 hash function. The size of the salt value is the same size as the hash function output. All other algorithm parameters use the defaults specified in Appendix A.2.3 of RFC 3447.

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

The RSASSA-PSS SHA-256 digital signature is generated as follows: generate a digital signature of the JWS Signing Input using RSASSA-PSS-SIGN, the SHA-256 hash function, and the MGF1 mask generation function with SHA-256 with the desired private key. This is the JWS Signature value.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWS Signature is a digital signature value computed using the corresponding algorithm:

"alg" Param Value	Digital Signature Algorithm
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384
PS512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512

The RSASSA-PSS SHA-256 digital signature for a JWS is validated as follows: submit the JWS Signing Input, the JWS Signature, and the public key corresponding to the private key used by the signer to the RSASSA-PSS-VERIFY algorithm using SHA-256 as the hash function and using MGF1 as the mask generation function with SHA-256.

Signing and validation with the RSASSA-PSS SHA-384 and RSASSA-PSS SHA-512 algorithms is performed identically to the procedure for RSASSA-PSS SHA-256 -- just using the alternative hash algorithm in both roles.

3.6. Using the Algorithm "none"

JWSs MAY also be created that do not provide integrity protection. Such a JWS is called an Unsecured JWS. An Unsecured JWS uses the "alg" value "none" and is formatted identically to other JWSs, but MUST use the empty octet sequence as its JWS Signature value. Recipients MUST verify that the JWS Signature value is the empty octet sequence.

Implementations that support Unsecured JWSs MUST NOT accept such objects as valid unless the application specifies that it is acceptable for a specific object to not be integrity protected. Implementations MUST NOT accept Unsecured JWSs by default. In order to mitigate downgrade attacks, applications MUST NOT signal acceptance of Unsecured JWSs at a global level, and SHOULD signal acceptance on a per-object basis. See Section 8.5 for security considerations associated with using this algorithm.

4. Cryptographic Algorithms for Key Management

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

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 CEK, producing the JWE Encrypted Key, or to use key agreement to agree upon the CEK.

"alg" Param Value	Key Management Algorithm	More Header Params	Implementation Requirements
RSA1_5	RSAES-PKCS1-v1_5	(none)	Recommended-
RSA-OAEP	RSAES OAEP using default parameters	(none)	Recommended+
RSA-OAEP-256	RSAES OAEP using SHA-256 and MGF1 with SHA-256	(none)	Optional
A128KW	AES Key Wrap with default initial value using 128-bit key	(none)	Recommended
A192KW	AES Key Wrap with default initial value using 192-bit key	(none)	Optional
A256KW	AES Key Wrap with default initial value using 256-bit key	(none)	Recommended
dir	Direct use of a shared symmetric key as the CEK	(none)	Recommended
ECDH-ES	Elliptic Curve Diffie-Hellman Ephemeral Static key agreement using Concat KDF	"epk", "apu", "apv"	Recommended+
ECDH-ES+A128KW	ECDH-ES using Concat KDF and CEK wrapped with "A128KW"	"epk", "apu", "apv"	Recommended
ECDH-ES+A192KW	ECDH-ES using Concat KDF and CEK wrapped with "A192KW"	"epk", "apu", "apv"	Optional

ECDH-ES+A256KW	ECDH-ES using Concat KDF and CEK wrapped with "A256KW"	"epk", "apu", "apv"	Recommended
A128GCMKW	Key wrapping with AES GCM using 128-bit key	"iv", "tag"	Optional
A192GCMKW	Key wrapping with AES GCM using 192-bit key	"iv", "tag"	Optional
A256GCMKW	Key wrapping with AES GCM using 256-bit key	"iv", "tag"	Optional
PBES2-HS256+A128KW	PBES2 with HMAC SHA-256 and "A128KW" wrapping	"p2s", "p2c"	Optional
PBES2-HS384+A192KW	PBES2 with HMAC SHA-384 and "A192KW" wrapping	"p2s", "p2c"	Optional
PBES2-HS512+A256KW	PBES2 with HMAC SHA-512 and "A256KW" wrapping	"p2s", "p2c"	Optional

The More Header Params column indicates what additional Header Parameters are used by the algorithm, beyond "alg", which all use. All but "dir" and "ECDH-ES" also produce a JWE Encrypted Key value.

The use of "+" in the Implementation Requirements column indicates that the requirement strength is likely to be increased in a future version of the specification. The use of "-" indicates that the requirement strength is likely to be decreased in a future version of the specification.

See Appendix A.2 for a table cross-referencing the JWE "alg" (algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

4.2. Key Encryption with RSAES-PKCS1-v1_5

This section defines the specifics of encrypting a JWE CEK with RSAES-PKCS1-v1_5 [RFC3447]. The "alg" (algorithm) Header Parameter value "RSA1_5" is used for this algorithm.

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

An example using this algorithm is shown in Appendix A.2 of [JWE].

4.3. Key Encryption with RSAES OAEP

This section defines the specifics of encrypting a JWE CEK with RSAES using Optimal Asymmetric Encryption Padding (OAEP) [RFC3447]. Two sets of parameters for using OAEP are defined, which use different hash functions. In the first case, the default parameters specified in Appendix A.2.1 of RFC 3447 are used. (Those default parameters are the SHA-1 hash function and the MGF1 with SHA-1 mask generation function.) In the second case, the SHA-256 hash function and the MGF1 with SHA-256 mask generation function are used.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWE Encrypted Key is the result of encrypting the CEK using the corresponding algorithm:

"alg" Param Value	Key Management Algorithm
RSA-OAEP	RSAES OAEP using default parameters
RSA-OAEP-256	RSAES OAEP using SHA-256 and MGF1 with SHA-256

A key of size 2048 bits or larger MUST be used with these algorithms. (This requirement is based on Table 4 (Security-strength time frames) of NIST SP 800-57 [NIST.800-57], which requires 112 bits of security for new uses, and Table 2 (Comparable strengths) of the same, which states that 2048-bit RSA keys provide 112 bits of security.)

An example using RSAES OAEP with the default parameters is shown in Appendix A.1 of [JWE].

4.4. Key Wrapping with AES Key Wrap

This section defines the specifics of encrypting a JWE CEK with the Advanced Encryption Standard (AES) Key Wrap Algorithm [RFC3394] using the default initial value specified in Section 2.2.3.1 of that document.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWE Encrypted Key is the result of encrypting the CEK using the corresponding algorithm and key size:

"alg" Param Value	Key Management Algorithm
A128KW	AES Key Wrap with default initial value using 128-bit key
A192KW	AES Key Wrap with default initial value using 192-bit key
A256KW	AES Key Wrap with default initial value using 256-bit key

An example using this algorithm is shown in Appendix A.3 of [JWE].

4.5. Direct Encryption with a Shared Symmetric Key

This section defines the specifics of directly performing symmetric key encryption without performing a key wrapping step. In this case, the shared symmetric key is used directly as the Content Encryption Key (CEK) value for the "enc" algorithm. An empty octet sequence is used as the JWE Encrypted Key value. The "alg" (algorithm) Header Parameter value "dir" is used in this case.

Refer to the security considerations on key lifetimes in Section 8.2 and AES GCM in Section 8.4 when considering utilizing direct encryption.

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

This section defines the specifics of key agreement with Elliptic Curve Diffie-Hellman Ephemeral Static [RFC6090], in combination with the Concat KDF, as defined in Section 5.8.1 of [NIST.800-56A]. The key agreement result can be used in one of two ways:

1. directly as the Content Encryption Key (CEK) for the "enc" algorithm, in the Direct Key Agreement mode, or

2. as a symmetric key used to wrap the CEK with the "A128KW", "A192KW", or "A256KW" algorithms, in the Key Agreement with Key Wrapping mode.

A new ephemeral public key value MUST be generated for each key agreement operation.

In Direct Key Agreement mode, the output of the Concat KDF MUST be a key of the same length as that used by the "enc" algorithm. In this case, the empty octet sequence is used as the JWE Encrypted Key value. The "alg" (algorithm) Header Parameter value "ECDH-ES" is used in the Direct Key Agreement mode.

In Key Agreement with Key Wrapping mode, the output of the Concat KDF MUST be a key of the length needed for the specified key wrapping algorithm. In this case, the JWE Encrypted Key is the CEK wrapped with the agreed-upon key.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWE Encrypted Key is the result of encrypting the CEK using the result of the key agreement algorithm as the key encryption key for the corresponding key wrapping algorithm:

"alg" Param Value	Key Management Algorithm
ECDH-ES+A128KW	ECDH-ES using Concat KDF and CEK wrapped with "A128KW"
ECDH-ES+A192KW	ECDH-ES using Concat KDF and CEK wrapped with "A192KW"
ECDH-ES+A256KW	ECDH-ES using Concat KDF and CEK wrapped with "A256KW"

4.6.1. Header Parameters Used for ECDH Key Agreement

The following Header Parameter names are used for key agreement as defined below.

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

The "epk" (ephemeral public key) value created by the originator for the use in key agreement algorithms. This key is represented as a JSON Web Key [JWK] public key value. It MUST contain only public key parameters and SHOULD contain only the minimum JWK parameters necessary to represent the key; other JWK parameters included can be checked for consistency and honored, or they can be ignored. This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

4.6.1.2. "apu" (Agreement PartyUInfo) Header Parameter

The "apu" (agreement PartyUInfo) value for key agreement algorithms using it (such as "ECDH-ES"), represented as a base64url-encoded string. When used, the PartyUInfo value contains information about the producer. Use of this Header Parameter is OPTIONAL. This Header Parameter MUST be understood and processed by implementations when these algorithms are used.

4.6.1.3. "apv" (Agreement PartyVInfo) Header Parameter

The "apv" (agreement PartyVInfo) value for key agreement algorithms using it (such as "ECDH-ES"), represented as a base64url encoded string. When used, the PartyVInfo value contains information about the recipient. Use of this Header Parameter is OPTIONAL. This Header Parameter MUST be understood and processed by implementations when these algorithms are used.

4.6.2. Key Derivation for ECDH Key Agreement

The key derivation process derives the agreed-upon key from the shared secret Z established through the ECDH algorithm, per Section 6.2.2.2 of [NIST.800-56A].

Key derivation is performed using the Concat KDF, as defined in Section 5.8.1 of [NIST.800-56A], where the Digest Method is SHA-256. The Concat KDF parameters are set as follows:

Z
: This is set to the representation of the shared secret Z as an octet sequence.

keydatalen
: This is set to the number of bits in the desired output key. For "ECDH-ES", this is length of the key used by the "enc" algorithm. For "ECDH-ES+A128KW", "ECDH-ES+A192KW", and "ECDH-ES+A256KW", this is 128, 192, and 256, respectively.

AlgorithmID
: The AlgorithmID value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data. In the Direct Key Agreement case, Data is set to the octets of the ASCII representation of the "enc" Header Parameter value. In the Key Agreement with Key Wrapping case, Data is set to the octets of the ASCII representation of the "alg" (algorithm) Header Parameter value.

PartyUInfo
: The PartyUInfo value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data. If an "apu" (agreement PartyUInfo) Header Parameter is present, Data is set to the result of base64url decoding the "apu" value and Datalen is set to the number of octets in Data. Otherwise, Datalen is set to 0 and Data is set to the empty octet sequence.

PartyVInfo
: The PartyVInfo value is of the form Datalen || Data, where Data is a variable-length string of zero or more octets, and Datalen is a fixed-length, big-endian 32-bit counter that indicates the length (in octets) of Data. If an "apv" (agreement PartyVInfo) Header Parameter is present, Data is set to the result of base64url decoding the "apv" value and Datalen is set to the number of octets in Data. Otherwise, Datalen is set to 0 and Data is set to the empty octet sequence.

SuppPubInfo
: This is set to the keydatalen represented as a 32-bit big-endian integer.

SuppPrivInfo
: This is set to the empty octet sequence.

Applications need to specify how the "apu" and "apv" Header Parameters are used for that application. The "apu" and "apv" values MUST be distinct, when used. Applications wishing to conform to [NIST.800-56A] need to provide values that meet the requirements of that document, e.g., by using values that identify the producer and consumer. Alternatively, applications MAY conduct key derivation in a manner similar to "Diffie-Hellman Key Agreement Method" [RFC2631]: in that case, the "apu" parameter MAY either be omitted or represent a random 512-bit value (analogous to PartyAInfo in Ephemeral-Static mode in RFC 2631) and the "apv" parameter SHOULD NOT be present.

See Appendix C for an example key agreement computation using this method.

4.7. Key Encryption with AES GCM

This section defines the specifics of encrypting a JWE Content Encryption Key (CEK) with Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) ([AES] and [NIST.800-38D]).

Use of an Initialization Vector (IV) of size 96 bits is REQUIRED with this algorithm. The IV is represented in base64url-encoded form as the "iv" (initialization vector) Header Parameter value.

The Additional Authenticated Data value used is the empty octet string.

The requested size of the Authentication Tag output MUST be 128 bits, regardless of the key size.

The JWE Encrypted Key value is the ciphertext output.

The Authentication Tag output is represented in base64url-encoded form as the "tag" (authentication tag) Header Parameter value.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWE Encrypted Key is the result of encrypting the CEK using the corresponding algorithm and key size:

"alg" Param Value	Key Management Algorithm
A128GCMKW	Key wrapping with AES GCM using 128-bit key
A192GCMKW	Key wrapping with AES GCM using 192-bit key
A256GCMKW	Key wrapping with AES GCM using 256-bit key

4.7.1. Header Parameters Used for AES GCM Key Encryption

The following Header Parameters are used for AES GCM key encryption.

4.7.1.1. "iv" (Initialization Vector) Header Parameter

The "iv" (initialization vector) Header Parameter value is the base64url-encoded representation of the 96-bit IV value used for the key encryption operation. This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

4.7.1.2. "tag" (Authentication Tag) Header Parameter

The "tag" (authentication tag) Header Parameter value is the base64url-encoded representation of the 128-bit Authentication Tag value resulting from the key encryption operation. This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

4.8. Key Encryption with PBES2

This section defines the specifics of performing password-based encryption of a JWE CEK, by first deriving a key encryption key from a user-supplied password using PBES2 schemes as specified in Section 6.2 of [RFC2898], then by encrypting the JWE CEK using the derived key.

These algorithms use HMAC SHA-2 algorithms as the Pseudorandom Function (PRF) for the PBKDF2 key derivation and AES Key Wrap [RFC3394] for the encryption scheme. The PBES2 password input is an octet sequence; if the password to be used is represented as a text string rather than an octet sequence, the UTF-8 encoding of the text string MUST be used as the octet sequence. The salt parameter MUST be computed from the "p2s" (PBES2 salt input) Header Parameter value and the "alg" (algorithm) Header Parameter value as specified in the "p2s" definition below. The iteration count parameter MUST be provided as the "p2c" (PBES2 count) Header Parameter value. The algorithms respectively use HMAC SHA-256, HMAC SHA-384, and HMAC SHA-512 as the PRF and use 128-, 192-, and 256-bit AES Key Wrap keys. Their derived-key lengths respectively are 16, 24, and 32 octets.

The following "alg" (algorithm) Header Parameter values are used to indicate that the JWE Encrypted Key is the result of encrypting the CEK using the result of the corresponding password-based encryption algorithm as the key encryption key for the corresponding key wrapping algorithm:

"alg" Param Value	Key Management Algorithm
PBES2-HS256+A128KW	PBES2 with HMAC SHA-256 and "A128KW" wrapping
PBES2-HS384+A192KW	PBES2 with HMAC SHA-384 and "A192KW" wrapping
PBES2-HS512+A256KW	PBES2 with HMAC SHA-512 and "A256KW" wrapping

See Appendix C of the JWK specification [JWK] for an example key encryption computation using "PBES2-HS256+A128KW".

4.8.1. Header Parameters Used for PBES2 Key Encryption

The following Header Parameters are used for Key Encryption with PBES2.

4.8.1.1. "p2s" (PBES2 Salt Input) Header Parameter

The "p2s" (PBES2 salt input) Header Parameter encodes a Salt Input value, which is used as part of the PBKDF2 salt value. The "p2s" value is BASE64URL(Salt Input). This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

The salt expands the possible keys that can be derived from a given password. A Salt Input value containing 8 or more octets MUST be used. A new Salt Input value MUST be generated randomly for every encryption operation; see RFC 4086 [RFC4086] for considerations on generating random values. The salt value used is (UTF8(Alg) || 0x00 || Salt Input), where Alg is the "alg" (algorithm) Header Parameter value.

4.8.1.2. "p2c" (PBES2 Count) Header Parameter

The "p2c" (PBES2 count) Header Parameter contains the PBKDF2 iteration count, represented as a positive JSON integer. This Header Parameter MUST be present and MUST be understood and processed by implementations when these algorithms are used.

The iteration count adds computational expense, ideally compounded by the possible range of keys introduced by the salt. A minimum iteration count of 1000 is RECOMMENDED.

5. Cryptographic Algorithms for Content Encryption

JWE uses cryptographic algorithms to encrypt and integrity-protect the plaintext and to integrity-protect the Additional Authenticated Data.

5.1. "enc" (Encryption Algorithm) Header Parameter Values for JWE

The table below is the set of "enc" (encryption algorithm) Header Parameter values that are defined by this specification for use with JWE.

"enc" Param Value	Content Encryption Algorithm	Implementation Requirements
A128CBC-HS256	AES_128_CBC_HMAC_SHA_256 authenticated encryption algorithm, as defined in Section 5.2.3	Required
A192CBC-HS384	AES_192_CBC_HMAC_SHA_384 authenticated encryption algorithm, as defined in Section 5.2.4	Optional
A256CBC-HS512	AES_256_CBC_HMAC_SHA_512 authenticated encryption algorithm, as defined in Section 5.2.5	Required
A128GCM	AES GCM using 128-bit key	Recommended
A192GCM	AES GCM using 192-bit key	Optional
A256GCM	AES GCM using 256-bit key	Recommended

All also use a JWE Initialization Vector value and produce JWE Ciphertext and JWE Authentication Tag values.

See Appendix A.3 for a table cross-referencing the JWE "enc" (encryption algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

5.2. AES_CBC_HMAC_SHA2 Algorithms

This section defines a family of authenticated encryption algorithms built using a composition of AES [AES] in Cipher Block Chaining (CBC) mode [NIST.800-38A] with PKCS #7 padding operations per Section 6.3 of [RFC5652] and HMAC ([RFC2104] and [SHS]) operations. This algorithm family is called AES_CBC_HMAC_SHA2. It also defines three instances of this family: the first using 128-bit CBC keys and HMAC SHA-256, the second using 192-bit CBC keys and HMAC SHA-384, and the third using 256-bit CBC keys and HMAC SHA-512. Test cases for these algorithms can be found in Appendix B.

These algorithms are based upon "Authenticated Encryption with AES-CBC and HMAC-SHA" [AEAD-CBC-SHA], performing the same cryptographic computations, but with the Initialization Vector (IV) and Authentication Tag values remaining separate, rather than being concatenated with the ciphertext value in the output representation. This option is discussed in Appendix B of that specification. This algorithm family is a generalization of the algorithm family in [AEAD-CBC-SHA] and can be used to implement those algorithms.

5.2.1. Conventions Used in Defining AES_CBC_HMAC_SHA2

We use the following notational conventions.

CBC-PKCS7-ENC(X, P) denotes the AES-CBC encryption of P using PKCS #7 padding utilizing the cipher with the key X.
MAC(Y, M) denotes the application of the MAC to the message M using the key Y.

5.2.2. Generic AES_CBC_HMAC_SHA2 Algorithm

This section defines AES_CBC_HMAC_SHA2 in a manner that is independent of the AES-CBC key size or hash function to be used. Sections 5.2.2.1 and 5.2.2.2 define the generic encryption and decryption algorithms. Sections 5.2.3 through 5.2.5 define instances of AES_CBC_HMAC_SHA2 that specify those details.

5.2.2.1. AES_CBC_HMAC_SHA2 Encryption

The authenticated encryption algorithm takes as input four octet strings: a secret key K, a plaintext P, Additional Authenticated Data A, and an Initialization Vector IV. The authenticated ciphertext value E and the Authentication Tag value T are provided as outputs. The data in the plaintext are encrypted and authenticated, and the Additional Authenticated Data are authenticated, but not encrypted.

The encryption process is as follows, or uses an equivalent set of steps:

1. The secondary keys MAC_KEY and ENC_KEY are generated from the input key K as follows. Each of these two keys is an octet string.

   MAC_KEY consists of the initial MAC_KEY_LEN octets of K, in order.
   ENC_KEY consists of the final ENC_KEY_LEN octets of K, in order.

The number of octets in the input key K MUST be the sum of MAC_KEY_LEN and ENC_KEY_LEN. The values of these parameters are specified by the Authenticated Encryption algorithms in Sections 5.2.3 through 5.2.5. Note that the MAC key comes before the encryption key in the input key K; this is in the opposite order of the algorithm names in the identifier "AES_CBC_HMAC_SHA2".

2. The IV used is a 128-bit value generated randomly or pseudorandomly for use in the cipher.

3. The plaintext is CBC encrypted using PKCS #7 padding using ENC_KEY as the key and the IV. We denote the ciphertext output from this step as E.

4. The octet string AL is equal to the number of bits in the Additional Authenticated Data A expressed as a 64-bit unsigned big-endian integer.

5. A message Authentication Tag T is computed by applying HMAC [RFC2104] to the following data, in order:

   the Additional Authenticated Data A,
   the Initialization Vector IV,
   the ciphertext E computed in the previous step, and
   the octet string AL defined above.

   The string MAC_KEY is used as the MAC key. We denote the output of the MAC computed in this step as M. The first T_LEN octets of M are used as T.

6. The ciphertext E and the Authentication Tag T are returned as the outputs of the authenticated encryption.

The encryption process can be illustrated as follows. Here K, P, A, IV, and E denote the key, plaintext, Additional Authenticated Data, Initialization Vector, and ciphertext, respectively.

```
MAC_KEY = initial MAC_KEY_LEN octets of K,
ENC_KEY = final ENC_KEY_LEN octets of K,
E = CBC-PKCS7-ENC(ENC_KEY, P),
M = MAC(MAC_KEY, A || IV || E || AL),
T = initial T_LEN octets of M.
```

5.2.2.2. AES_CBC_HMAC_SHA2 Decryption

The authenticated decryption operation has five inputs: K, A, IV, E, and T as defined above. It has only a single output: either a plaintext value P or a special symbol FAIL that indicates that the inputs are not authentic. The authenticated decryption algorithm is as follows, or uses an equivalent set of steps:

1. The secondary keys MAC_KEY and ENC_KEY are generated from the input key K as in Step 1 of Section 5.2.2.1.

2. The integrity and authenticity of A and E are checked by computing an HMAC with the inputs as in Step 5 of Section 5.2.2.1. The value T, from the previous step, is compared to the first MAC_KEY length bits of the HMAC output. If those values are identical, then A and E are considered valid, and processing is continued. Otherwise, all of the data used in the MAC validation are discarded, and the authenticated decryption operation returns an indication that it failed, and the operation halts. (But see Section 11.5 of [JWE] for security considerations on thwarting timing attacks.)

3. The value E is decrypted and the PKCS #7 padding is checked and removed. The value IV is used as the Initialization Vector. The value ENC_KEY is used as the decryption key.

4. The plaintext value is returned.

5.2.3. AES_128_CBC_HMAC_SHA_256

This algorithm is a concrete instantiation of the generic AES_CBC_HMAC_SHA2 algorithm above. It uses the HMAC message authentication code [RFC2104] with the SHA-256 hash function [SHS] to provide message authentication, with the HMAC output truncated to 128 bits, corresponding to the HMAC-SHA-256-128 algorithm defined in [RFC4868]. For encryption, it uses AES in the CBC mode of operation as defined in Section 6.2 of [NIST.800-38A], with PKCS #7 padding and a 128-bit IV value.

The AES_CBC_HMAC_SHA2 parameters specific to AES_128_CBC_HMAC_SHA_256 are:

> The input key K is 32 octets long.
> ENC_KEY_LEN is 16 octets.
> MAC_KEY_LEN is 16 octets.
> The SHA-256 hash algorithm is used for the HMAC.
> The HMAC-SHA-256 output is truncated to T_LEN=16 octets, by stripping off the final 16 octets.

5.2.4. AES_192_CBC_HMAC_SHA_384

AES_192_CBC_HMAC_SHA_384 is based on AES_128_CBC_HMAC_SHA_256, but with the following differences:

- The input key K is 48 octets long instead of 32.
- ENC_KEY_LEN is 24 octets instead of 16.
- MAC_KEY_LEN is 24 octets instead of 16.
- SHA-384 is used for the HMAC instead of SHA-256.
- The HMAC SHA-384 value is truncated to T_LEN=24 octets instead of 16.

5.2.5. AES_256_CBC_HMAC_SHA_512

AES_256_CBC_HMAC_SHA_512 is based on AES_128_CBC_HMAC_SHA_256, but with the following differences:

- The input key K is 64 octets long instead of 32.
- ENC_KEY_LEN is 32 octets instead of 16.
- MAC_KEY_LEN is 32 octets instead of 16.
- SHA-512 is used for the HMAC instead of SHA-256.
- The HMAC SHA-512 value is truncated to T_LEN=32 octets instead of 16.

5.2.6. Content Encryption with AES_CBC_HMAC_SHA2

This section defines the specifics of performing authenticated encryption with the AES_CBC_HMAC_SHA2 algorithms.

The CEK is used as the secret key K.

The following "enc" (encryption algorithm) Header Parameter values are used to indicate that the JWE Ciphertext and JWE Authentication Tag values have been computed using the corresponding algorithm:

"enc" Param Value	Content Encryption Algorithm
A128CBC-HS256	AES_128_CBC_HMAC_SHA_256 authenticated encryption algorithm, as defined in Section 5.2.3
A192CBC-HS384	AES_192_CBC_HMAC_SHA_384 authenticated encryption algorithm, as defined in Section 5.2.4
A256CBC-HS512	AES_256_CBC_HMAC_SHA_512 authenticated encryption algorithm, as defined in Section 5.2.5

5.3. Content Encryption with AES GCM

This section defines the specifics of performing authenticated encryption with AES in Galois/Counter Mode (GCM) ([AES] and [NIST.800-38D]).

The CEK is used as the encryption key.

Use of an IV of size 96 bits is REQUIRED with this algorithm.

The requested size of the Authentication Tag output MUST be 128 bits, regardless of the key size.

The following "enc" (encryption algorithm) Header Parameter values are used to indicate that the JWE Ciphertext and JWE Authentication Tag values have been computed using the corresponding algorithm and key size:

"enc" Param Value	Content Encryption Algorithm
A128GCM	AES GCM using 128-bit key
A192GCM	AES GCM using 192-bit key
A256GCM	AES GCM using 256-bit key

An example using this algorithm is shown in Appendix A.1 of [JWE].

6. Cryptographic Algorithms for Keys

A JSON Web Key (JWK) [JWK] is a JSON data structure that represents a cryptographic key. These keys can be either asymmetric or symmetric. They can hold both public and private information about the key. This section defines the parameters for keys using the algorithms specified by this document.

6.1. "kty" (Key Type) Parameter Values

The table below is the set of "kty" (key type) parameter values that are defined by this specification for use in JWKs.

"kty" Param Value	Key Type	Implementation Requirements
EC	Elliptic Curve [DSS]	Recommended+
RSA	RSA [RFC3447]	Required
oct	Octet sequence (used to represent symmetric keys)	Required

The use of "+" in the Implementation Requirements column indicates that the requirement strength is likely to be increased in a future version of the specification.

6.2. Parameters for Elliptic Curve Keys

JWKs can represent Elliptic Curve [DSS] keys. In this case, the "kty" member value is "EC".

6.2.1. Parameters for Elliptic Curve Public Keys

An Elliptic Curve public key is represented by a pair of coordinates drawn from a finite field, which together define a point on an Elliptic Curve. The following members MUST be present for all Elliptic Curve public keys:

- "crv"
- "x"

The following member MUST also be present for Elliptic Curve public keys for the three curves defined in the following section:

- "y"

6.2.1.1. "crv" (Curve) Parameter

The "crv" (curve) parameter identifies the cryptographic curve used with the key. Curve values from [DSS] used by this specification are:

- "P-256"
- "P-384"
- "P-521"

These values are registered in the IANA "JSON Web Key Elliptic Curve" registry defined in Section 7.6. Additional "crv" values can be registered by other specifications. Specifications registering additional curves must define what parameters are used to represent keys for the curves registered. The "crv" value is a case-sensitive string.

SEC1 [SEC1] point compression is not supported for any of these three curves.

6.2.1.2. "x" (X Coordinate) Parameter

The "x" (x coordinate) parameter contains the x coordinate for the Elliptic Curve point. It is represented as the base64url encoding of the octet string representation of the coordinate, as defined in Section 2.3.5 of SEC1 [SEC1]. The length of this octet string MUST be the full size of a coordinate for the curve specified in the "crv" parameter. For example, if the value of "crv" is "P-521", the octet string must be 66 octets long.

6.2.1.3. "y" (Y Coordinate) Parameter

The "y" (y coordinate) parameter contains the y coordinate for the Elliptic Curve point. It is represented as the base64url encoding of the octet string representation of the coordinate, as defined in Section 2.3.5 of SEC1 [SEC1]. The length of this octet string MUST be the full size of a coordinate for the curve specified in the "crv" parameter. For example, if the value of "crv" is "P-521", the octet string must be 66 octets long.

6.2.2. Parameters for Elliptic Curve Private Keys

In addition to the members used to represent Elliptic Curve public keys, the following member MUST be present to represent Elliptic Curve private keys.

6.2.2.1. "d" (ECC Private Key) Parameter

The "d" (ECC private key) parameter contains the Elliptic Curve private key value. It is represented as the base64url encoding of the octet string representation of the private key value, as defined in Section 2.3.7 of SEC1 [SEC1]. The length of this octet string MUST be ceiling(log-base-2(n)/8) octets (where n is the order of the curve).

6.3. Parameters for RSA Keys

JWKs can represent RSA [RFC3447] keys. In this case, the "kty" member value is "RSA". The semantics of the parameters defined below are the same as those defined in Sections 3.1 and 3.2 of RFC 3447.

6.3.1. Parameters for RSA Public Keys

The following members MUST be present for RSA public keys.

6.3.1.1. "n" (Modulus) Parameter

The "n" (modulus) parameter contains the modulus value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

Note that implementers have found that some cryptographic libraries prefix an extra zero-valued octet to the modulus representations they return, for instance, returning 257 octets for a 2048-bit key, rather than 256. Implementations using such libraries will need to take care to omit the extra octet from the base64url-encoded representation.

6.3.1.2. "e" (Exponent) Parameter

The "e" (exponent) parameter contains the exponent value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

For instance, when representing the value 65537, the octet sequence to be base64url-encoded MUST consist of the three octets [1, 0, 1]; the resulting representation for this value is "AQAB".

6.3.2. Parameters for RSA Private Keys

In addition to the members used to represent RSA public keys, the following members are used to represent RSA private keys. The parameter "d" is REQUIRED for RSA private keys. The others enable optimizations and SHOULD be included by producers of JWKs representing RSA private keys. If the producer includes any of the other private key parameters, then all of the others MUST be present, with the exception of "oth", which MUST only be present when more than two prime factors were used.

6.3.2.1. "d" (Private Exponent) Parameter

The "d" (private exponent) parameter contains the private exponent value for the RSA private key. It is represented as a Base64urlUInt-encoded value.

6.3.2.2. "p" (First Prime Factor) Parameter

The "p" (first prime factor) parameter contains the first prime factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.3. "q" (Second Prime Factor) Parameter

The "q" (second prime factor) parameter contains the second prime factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.4. "dp" (First Factor CRT Exponent) Parameter

The "dp" (first factor CRT exponent) parameter contains the Chinese Remainder Theorem (CRT) exponent of the first factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.5. "dq" (Second Factor CRT Exponent) Parameter

The "dq" (second factor CRT exponent) parameter contains the CRT exponent of the second factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.6. "qi" (First CRT Coefficient) Parameter

The "qi" (first CRT coefficient) parameter contains the CRT coefficient of the second factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.7. "oth" (Other Primes Info) Parameter

The "oth" (other primes info) parameter contains an array of information about any third and subsequent primes, should they exist. When only two primes have been used (the normal case), this parameter MUST be omitted. When three or more primes have been used, the number of array elements MUST be the number of primes used minus two. For more information on this case, see the description of the OtherPrimeInfo parameters in Appendix A.1.2 of RFC 3447 [RFC3447], upon which the following parameters are modeled. If the consumer of a JWK does not support private keys with more than two primes and it encounters a private key that includes the "oth" parameter, then it MUST NOT use the key. Each array element MUST be an object with the following members.

6.3.2.7.1. "r" (Prime Factor)

The "r" (prime factor) parameter within an "oth" array member represents the value of a subsequent prime factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.7.2. "d" (Factor CRT Exponent)

The "d" (factor CRT exponent) parameter within an "oth" array member represents the CRT exponent of the corresponding prime factor. It is represented as a Base64urlUInt-encoded value.

6.3.2.7.3. "t" (Factor CRT Coefficient)

The "t" (factor CRT coefficient) parameter within an "oth" array member represents the CRT coefficient of the corresponding prime factor. It is represented as a Base64urlUInt-encoded value.

6.4. Parameters for Symmetric Keys

When the JWK "kty" member value is "oct" (octet sequence), the member "k" (see Section 6.4.1) is used to represent a symmetric key (or another key whose value is a single octet sequence). An "alg" member SHOULD also be present to identify the algorithm intended to be used with the key, unless the application uses another means or convention to determine the algorithm used.

6.4.1. "k" (Key Value) Parameter

The "k" (key value) parameter contains the value of the symmetric (or other single-valued) key. It is represented as the base64url encoding of the octet sequence containing the key value.

7. IANA Considerations

The following registration procedure is used for all the registries established by this specification.

The registration procedure for values is Specification Required [RFC5226] after a three-week review period on the jose-reg-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register algorithm: example").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@ietf.org mailing list) for resolution.

Criteria that should be applied by the Designated Experts include determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or useful only for a single application, and whether the registration description is clear.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

7.1. 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) and "enc" (encryption algorithm) Header Parameters. The registry records the algorithm name, the algorithm description, the algorithm usage locations, the implementation requirements, the change controller, and a reference to the specification that defines it. The same algorithm name can be registered multiple times, provided that the sets of usage locations are disjoint.

It is suggested that the length of the key be included in the algorithm name when multiple variations of algorithms are being registered that use keys of different lengths and the key lengths for each need to be fixed (for instance, because they will be created by key derivation functions). This allows readers of the JSON text to more easily make security decisions.

The Designated Experts should perform reasonable due diligence that algorithms being registered either are currently considered cryptographically credible or are being registered as Deprecated or Prohibited.

The implementation requirements of an algorithm may be changed over time as the cryptographic landscape evolves, for instance, to change the status of an algorithm to Deprecated or to change the status of an algorithm from Optional to Recommended+ or Required. Changes of implementation requirements are only permitted on a Specification Required basis after review by the Designated Experts, with the new specification defining the revised implementation requirements level.

7.1.1. Registration Template

Algorithm Name:
: The name requested (e.g., "HS256"). This name is a case-sensitive ASCII string. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Algorithm Description:
: Brief description of the algorithm (e.g., "HMAC using SHA-256").

Algorithm Usage Location(s):
: The algorithm usage locations. This must be one or more of the values "alg" or "enc" if the algorithm is to be used with JWS or JWE. The value "JWK" is used if the algorithm identifier will be used as a JWK "alg" member value, but will not be used with JWS or JWE; this could be the case, for instance, for non-authenticated encryption algorithms. Other values may be used with the approval of a Designated Expert.

JOSE Implementation Requirements:
: The algorithm implementation requirements for JWS and JWE, which must be one the words Required, Recommended, Optional, Deprecated, or Prohibited. Optionally, the word can be followed by a "+" or "-". The use of "+" indicates that the requirement strength is likely to be increased in a future version of the specification. The use of "-" indicates that the requirement strength is likely to be decreased in a future version of the specification. Any identifiers registered for non-authenticated encryption algorithms or other algorithms that are otherwise unsuitable for direct use as JWS or JWE algorithms must be registered as "Prohibited".

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

Algorithm Analysis Documents(s):
References to a publication or publications in well-known cryptographic conferences, by national standards bodies, or by other authoritative sources analyzing the cryptographic soundness of the algorithm to be registered. The Designated Experts may require convincing evidence of the cryptographic soundness of a new algorithm to be provided with the registration request unless the algorithm is being registered as Deprecated or Prohibited. Having gone through working group and IETF review, the initial registrations made by this document are exempt from the need to provide this information.

7.1.2. Initial Registry Contents

- Algorithm Name: "HS256"
- Algorithm Description: HMAC using SHA-256
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Required
- Change Controller: IESG
- Specification Document(s): Section 3.2 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "HS384"
- Algorithm Description: HMAC using SHA-384
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.2 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "HS512"
- Algorithm Description: HMAC using SHA-512
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.2 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RS256"
- Algorithm Description: RSASSA-PKCS1-v1_5 using SHA-256
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 3.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RS384"
- Algorithm Description: RSASSA-PKCS1-v1_5 using SHA-384
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RS512"
- Algorithm Description: RSASSA-PKCS1-v1_5 using SHA-512
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ES256"
- Algorithm Description: ECDSA using P-256 and SHA-256
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended+
- Change Controller: IESG
- Specification Document(s): Section 3.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ES384"
- Algorithm Description: ECDSA using P-384 and SHA-384
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ES512"
- Algorithm Description: ECDSA using P-521 and SHA-512
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PS256"
- Algorithm Description: RSASSA-PSS using SHA-256 and MGF1 with SHA-256
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.5 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PS384"
- Algorithm Description: RSASSA-PSS using SHA-384 and MGF1 with SHA-384
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.5 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PS512"
- Algorithm Description: RSASSA-PSS using SHA-512 and MGF1 with SHA-512
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.5 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "none"
- Algorithm Description: No digital signature or MAC performed
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.6 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RSA1_5"
- Algorithm Description: RSAES-PKCS1-v1_5
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended-
- Change Controller: IESG
- Specification Document(s): Section 4.2 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RSA-OAEP"
- Algorithm Description: RSAES OAEP using default parameters
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended+
- Change Controller: IESG
- Specification Document(s): Section 4.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "RSA-OAEP-256"
- Algorithm Description: RSAES OAEP using SHA-256 and MGF1 with SHA-256
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A128KW"
- Algorithm Description: AES Key Wrap using 128-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 4.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A192KW"
- Algorithm Description: AES Key Wrap using 192-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A256KW"
- Algorithm Description: AES Key Wrap using 256-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 4.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "dir"
- Algorithm Description: Direct use of a shared symmetric key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 4.5 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ECDH-ES"
- Algorithm Description: ECDH-ES using Concat KDF
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended+
- Change Controller: IESG
- Specification Document(s): Section 4.6 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ECDH-ES+A128KW"
- Algorithm Description: ECDH-ES using Concat KDF and "A128KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 4.6 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ECDH-ES+A192KW"
- Algorithm Description: ECDH-ES using Concat KDF and "A192KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.6 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "ECDH-ES+A256KW"
- Algorithm Description: ECDH-ES using Concat KDF and "A256KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 4.6 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A128GCMKW"
- Algorithm Description: Key wrapping with AES GCM using 128-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.7 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A192GCMKW"
- Algorithm Description: Key wrapping with AES GCM using 192-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.7 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A256GCMKW"
- Algorithm Description: Key wrapping with AES GCM using 256-bit key
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.7 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PBES2-HS256+A128KW"
- Algorithm Description: PBES2 with HMAC SHA-256 and "A128KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.8 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PBES2-HS384+A192KW"
- Algorithm Description: PBES2 with HMAC SHA-384 and "A192KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.8 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "PBES2-HS512+A256KW"
- Algorithm Description: PBES2 with HMAC SHA-512 and "A256KW" wrapping
- Algorithm Usage Location(s): "alg"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 4.8 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A128CBC-HS256"
- Algorithm Description: AES_128_CBC_HMAC_SHA_256 authenticated encryption algorithm
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Required
- Change Controller: IESG
- Specification Document(s): Section 5.2.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A192CBC-HS384"
- Algorithm Description: AES_192_CBC_HMAC_SHA_384 authenticated encryption algorithm
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 5.2.4 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A256CBC-HS512"
- Algorithm Description: AES_256_CBC_HMAC_SHA_512 authenticated encryption algorithm
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Required
- Change Controller: IESG
- Specification Document(s): Section 5.2.5 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A128GCM"
- Algorithm Description: AES GCM using 128-bit key
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 5.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A192GCM"
- Algorithm Description: AES GCM using 192-bit key
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 5.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

- Algorithm Name: "A256GCM"
- Algorithm Description: AES GCM using 256-bit key
- Algorithm Usage Location(s): "enc"
- JOSE Implementation Requirements: Recommended
- Change Controller: IESG
- Specification Document(s): Section 5.3 of RFC 7518
- Algorithm Analysis Documents(s): n/a

7.2. Header Parameter Names Registration

This section registers the Header Parameter names defined in Sections 4.6.1, 4.7.1, and 4.8.1 of this specification in the IANA "JSON Web Signature and Encryption Header Parameters" registry established by [JWS].

7.2.1. Registry Contents

- Header Parameter Name: "epk"
- Header Parameter Description: Ephemeral Public Key
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.6.1.1 of RFC 7518

- Header Parameter Name: "apu"
- Header Parameter Description: Agreement PartyUInfo
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.6.1.2 of RFC 7518

- Header Parameter Name: "apv"
- Header Parameter Description: Agreement PartyVInfo
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.6.1.3 of RFC 7518

- Header Parameter Name: "iv"
- Header Parameter Description: Initialization Vector
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.7.1.1 of RFC 7518

- Header Parameter Name: "tag"
- Header Parameter Description: Authentication Tag
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.7.1.2 of RFC 7518

- Header Parameter Name: "p2s"
- Header Parameter Description: PBES2 Salt Input
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.8.1.1 of RFC 7518

- Header Parameter Name: "p2c"
- Header Parameter Description: PBES2 Count
- Header Parameter Usage Location(s): JWE
- Change Controller: IESG
- Specification Document(s): Section 4.8.1.2 of RFC 7518

7.3. JSON Web Encryption Compression Algorithms Registry

This specification establishes the IANA "JSON Web Encryption Compression Algorithms" registry for JWE "zip" member values. The registry records the compression algorithm value and a reference to the specification that defines it.

7.3.1. Registration Template

Compression Algorithm Value:
: The name requested (e.g., "DEF"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Compression Algorithm Description:
: Brief description of the compression algorithm (e.g., "DEFLATE").

Change Controller:
: For Standards Track RFCs, list "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

7.3.2. Initial Registry Contents

- Compression Algorithm Value: "DEF"
- Compression Algorithm Description: DEFLATE
- Change Controller: IESG
- Specification Document(s): JSON Web Encryption (JWE) [JWE]

7.4. JSON Web Key Types Registry

This specification establishes the IANA "JSON Web Key Types" registry for values of the JWK "kty" (key type) parameter. The registry records the "kty" value, implementation requirements, and a reference to the specification that defines it.

The implementation requirements of a key type may be changed over time as the cryptographic landscape evolves, for instance, to change the status of a key type to Deprecated or to change the status of a key type from Optional to Recommended+ or Required. Changes of implementation requirements are only permitted on a Specification Required basis after review by the Designated Experts, with the new specification defining the revised implementation requirements level.

7.4.1. Registration Template

"kty" Parameter Value:
: The name requested (e.g., "EC"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Key Type Description:
: Brief description of the Key Type (e.g., "Elliptic Curve").

Change Controller:
: For Standards Track RFCs, list "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

JOSE Implementation Requirements:
The key type implementation requirements for JWS and JWE, which must be one the words Required, Recommended, Optional, Deprecated, or Prohibited. Optionally, the word can be followed by a "+" or "-". The use of "+" indicates that the requirement strength is likely to be increased in a future version of the specification. The use of "-" indicates that the requirement strength is likely to be decreased in a future version of the specification.

Specification Document(s):
Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

7.4.2. Initial Registry Contents

This section registers the values defined in Section 6.1.

- "kty" Parameter Value: "EC"
- Key Type Description: Elliptic Curve
- JOSE Implementation Requirements: Recommended+
- Change Controller: IESG
- Specification Document(s): Section 6.2 of RFC 7518

- "kty" Parameter Value: "RSA"
- Key Type Description: RSA
- JOSE Implementation Requirements: Required
- Change Controller: IESG
- Specification Document(s): Section 6.3 of RFC 7518

- "kty" Parameter Value: "oct"
- Key Type Description: Octet Sequence
- JOSE Implementation Requirements: Required
- Change Controller: IESG
- Specification Document(s): Section 6.4 of RFC 7518

7.5. JSON Web Key Parameters Registration

This section registers the parameter names defined in Sections 6.2, 6.3, and 6.4 of this specification in the IANA "JSON Web Key Parameters" registry established by [JWK].

7.5.1. Registry Contents

- Parameter Name: "crv"
- Parameter Description: Curve
- Used with "kty" Value(s): "EC"
- Parameter Information Class: Public
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.1 of RFC 7518

- Parameter Name: "x"
- Parameter Description: X Coordinate
- Used with "kty" Value(s): "EC"
- Parameter Information Class: Public
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.2 of RFC 7518

- Parameter Name: "y"
- Parameter Description: Y Coordinate
- Used with "kty" Value(s): "EC"
- Parameter Information Class: Public
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.3 of RFC 7518

- Parameter Name: "d"
- Parameter Description: ECC Private Key
- Used with "kty" Value(s): "EC"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.2.2.1 of RFC 7518

- Parameter Name: "n"
- Parameter Description: Modulus
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Public
- Change Controller: IESG
- Specification Document(s): Section 6.3.1.1 of RFC 7518

- Parameter Name: "e"
- Parameter Description: Exponent
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Public
- Change Controller: IESG
- Specification Document(s): Section 6.3.1.2 of RFC 7518

- Parameter Name: "d"
- Parameter Description: Private Exponent
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.1 of RFC 7518

- Parameter Name: "p"
- Parameter Description: First Prime Factor
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.2 of RFC 7518

- Parameter Name: "q"
- Parameter Description: Second Prime Factor
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.3 of RFC 7518

- Parameter Name: "dp"
- Parameter Description: First Factor CRT Exponent
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.4 of RFC 7518

- Parameter Name: "dq"
- Parameter Description: Second Factor CRT Exponent
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.5 of RFC 7518

- Parameter Name: "qi"
- Parameter Description: First CRT Coefficient
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.6 of RFC 7518

- Parameter Name: "oth"
- Parameter Description: Other Primes Info
- Used with "kty" Value(s): "RSA"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.3.2.7 of RFC 7518

- Parameter Name: "k"
- Parameter Description: Key Value
- Used with "kty" Value(s): "oct"
- Parameter Information Class: Private
- Change Controller: IESG
- Specification Document(s): Section 6.4.1 of RFC 7518

7.6. JSON Web Key Elliptic Curve Registry

This section establishes the IANA "JSON Web Key Elliptic Curve" registry for JWK "crv" member values. The registry records the curve name, implementation requirements, and a reference to the specification that defines it. This specification registers the parameter names defined in Section 6.2.1.1.

The implementation requirements of a curve may be changed over time as the cryptographic landscape evolves, for instance, to change the status of a curve to Deprecated or to change the status of a curve from Optional to Recommended+ or Required. Changes of implementation requirements are only permitted on a Specification Required basis after review by the Designated Experts, with the new specification defining the revised implementation requirements level.

7.6.1. Registration Template

Curve Name:
: The name requested (e.g., "P-256"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Curve Description:
: Brief description of the curve (e.g., "P-256 Curve").

JOSE Implementation Requirements:
: The curve implementation requirements for JWS and JWE, which must be one the words Required, Recommended, Optional, Deprecated, or Prohibited. Optionally, the word can be followed by a "+" or "-".

The use of "+" indicates that the requirement strength is likely to be increased in a future version of the specification. The use of "-" indicates that the requirement strength is likely to be decreased in a future version of the specification.

Change Controller:
For Standards Track RFCs, list "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

7.6.2. Initial Registry Contents

- Curve Name: "P-256"
- Curve Description: P-256 Curve
- JOSE Implementation Requirements: Recommended+
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.1 of RFC 7518

- Curve Name: "P-384"
- Curve Description: P-384 Curve
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.1 of RFC 7518

- Curve Name: "P-521"
- Curve Description: P-521 Curve
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 6.2.1.1 of RFC 7518

8. Security Considerations

All of the security issues that are pertinent to any cryptographic application must be addressed by JWS/JWE/JWK agents. Among these issues are protecting the user's asymmetric private and symmetric secret keys and employing countermeasures to various attacks.

The security considerations in [AES], [DSS], [JWE], [JWK], [JWS], [NIST.800-38D], [NIST.800-56A], [NIST.800-107], [RFC2104], [RFC3394], [RFC3447], [RFC5116], [RFC6090], and [SHS] apply to this specification.

8.1. Cryptographic Agility

Implementers should be aware that cryptographic algorithms become weaker with time. As new cryptanalysis techniques are developed and computing performance improves, the work factor to break a particular cryptographic algorithm will be reduced. Therefore, implementers and deployments must be prepared for the set of algorithms that are supported and used to change over time. Thus, cryptographic algorithm implementations should be modular, allowing new algorithms to be readily inserted.

8.2. Key Lifetimes

Many algorithms have associated security considerations related to key lifetimes and/or the number of times that a key may be used. Those security considerations continue to apply when using those algorithms with JOSE data structures. See NIST SP 800-57 [NIST.800-57] for specific guidance on key lifetimes.

8.3. RSAES-PKCS1-v1_5 Security Considerations

While Section 8 of RFC 3447 [RFC3447] explicitly calls for people not to adopt RSASSA-PKCS1-v1_5 for new applications and instead requests that people transition to RSASSA-PSS, this specification does include RSASSA-PKCS1-v1_5, for interoperability reasons, because it is commonly implemented.

Keys used with RSAES-PKCS1-v1_5 must follow the constraints in Section 7.2 of RFC 3447. Also, keys with a low public key exponent value, as described in Section 3 of "Twenty Years of Attacks on the RSA Cryptosystem" [Boneh99], must not be used.

8.4. AES GCM Security Considerations

Keys used with AES GCM must follow the constraints in Section 8.3 of [NIST.800-38D], which states: "The total number of invocations of the authenticated encryption function shall not exceed 2^32, including all IV lengths and all instances of the authenticated encryption function with the given key". In accordance with this rule, AES GCM MUST NOT be used with the same key value more than 2^32 times.

An IV value MUST NOT ever be used multiple times with the same AES GCM key. One way to prevent this is to store a counter with the key and increment it with every use. The counter can also be used to prevent exceeding the 2^32 limit above.

This security consideration does not apply to the composite AES-CBC HMAC SHA-2 or AES Key Wrap algorithms.

8.5. Unsecured JWS Security Considerations

Unsecured JWSs (JWSs that use the "alg" value "none") provide no integrity protection. Thus, they must only be used in contexts in which the payload is secured by means other than a digital signature or MAC value, or they need not be secured.

An example means of preventing accepting Unsecured JWSs by default is for the "verify" method of a hypothetical JWS software library to have a Boolean "acceptUnsecured" parameter that indicates "none" is an acceptable "alg" value. As another example, the "verify" method might take a list of algorithms that are acceptable to the application as a parameter and would reject Unsecured JWS values if "none" is not in that list.

The following example illustrates the reasons for not accepting Unsecured JWSs at a global level. Suppose an application accepts JWSs over two channels, (1) HTTP and (2) HTTPS with client authentication. It requires a JWS Signature on objects received over HTTP, but accepts Unsecured JWSs over HTTPS. If the application were to globally indicate that "none" is acceptable, then an attacker could provide it with an Unsecured JWS over HTTP and still have that object successfully validate. Instead, the application needs to indicate acceptance of "none" for each object received over HTTPS (e.g., by setting "acceptUnsecured" to "true" for the first hypothetical JWS software library above), but not for each object received over HTTP.

8.6. Denial-of-Service Attacks

Receiving agents that validate signatures and sending agents that encrypt messages need to be cautious of cryptographic processing usage when validating signatures and encrypting messages using keys larger than those mandated in this specification. An attacker could supply content using keys that would result in excessive cryptographic processing, for example, keys larger than those mandated in this specification. Implementations should set and enforce upper limits on the key sizes they accept. Section 5.6.1 (Comparable Algorithm Strengths) of NIST SP 800-57 [NIST.800-57] contains statements on largest approved key sizes that may be applicable.

8.7. Reusing Key Material when Encrypting Keys

It is NOT RECOMMENDED to reuse the same entire set of key material (Key Encryption Key, Content Encryption Key, Initialization Vector, etc.) to encrypt multiple JWK or JWK Set objects, or to encrypt the same JWK or JWK Set object multiple times. One suggestion for

preventing reuse is to always generate at least one new piece of key material for each encryption operation (e.g., a new Content Encryption Key, a new IV, and/or a new PBES2 Salt), based on the considerations noted in this document as well as from RFC 4086 [RFC4086].

8.8. Password Considerations

Passwords are vulnerable to a number of attacks. To help mitigate some of these limitations, this document applies principles from RFC 2898 [RFC2898] to derive cryptographic keys from user-supplied passwords.

However, the strength of the password still has a significant impact. A high-entropy password has greater resistance to dictionary attacks. [NIST.800-63-2] contains guidelines for estimating password entropy, which can help applications and users generate stronger passwords.

An ideal password is one that is as large as (or larger than) the derived key length. However, passwords larger than a certain algorithm-specific size are first hashed, which reduces an attacker's effective search space to the length of the hash algorithm. It is RECOMMENDED that a password used for "PBES2-HS256+A128KW" be no shorter than 16 octets and no longer than 128 octets and a password used for "PBES2-HS512+A256KW" be no shorter than 32 octets and no longer than 128 octets long.

Still, care needs to be taken in where and how password-based encryption is used. These algorithms can still be susceptible to dictionary-based attacks if the iteration count is too small; this is of particular concern if these algorithms are used to protect data that an attacker can have indefinite number of attempts to circumvent the protection, such as protected data stored on a file system.

8.9. Key Entropy and Random Values

See Section 10.1 of [JWS] for security considerations on key entropy and random values.

8.10. Differences between Digital Signatures and MACs

See Section 10.5 of [JWS] for security considerations on differences between digital signatures and MACs.

8.11. Using Matching Algorithm Strengths

See Section 11.3 of [JWE] for security considerations on using matching algorithm strengths.

8.12. Adaptive Chosen-Ciphertext Attacks

See Section 11.4 of [JWE] for security considerations on adaptive chosen-ciphertext attacks.

8.13. Timing Attacks

See Section 10.9 of [JWS] and Section 11.5 of [JWE] for security considerations on timing attacks.

8.14. RSA Private Key Representations and Blinding

See Section 9.3 of [JWK] for security considerations on RSA private key representations and blinding.

9. Internationalization Considerations

Passwords obtained from users are likely to require preparation and normalization to account for differences of octet sequences generated by different input devices, locales, etc. It is RECOMMENDED that applications perform the steps outlined in [PRECIS] to prepare a password supplied directly by a user before performing key derivation and encryption.

10. References

10.1. Normative References

[AES] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[Boneh99] "Twenty Years of Attacks on the RSA Cryptosystem", Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203-213, 1999, <http://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf>.

[DSS] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS PUB 186-4, July 2013, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[NIST.800-38A] National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation", NIST Special Publication 800-38A, December 2001, <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>.

[NIST.800-38D] National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, December 2001, <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>.

[NIST.800-56A] National Institute of Standards and Technology (NIST), "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A, Revision 2, May 2013, <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf>.

[NIST.800-57] National Institute of Standards and Technology (NIST), "Recommendation for Key Management - Part 1: General (Revision 3)", NIST Special Publication 800-57, Part 1, Revision 3, July 2012, <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf>.

[RFC20] Cerf, V., "ASCII format for Network Interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <http://www.rfc-editor.org/info/rfc2898>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <http://www.rfc-editor.org/info/rfc3394>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <http://www.rfc-editor.org/info/rfc3447>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <http://www.rfc-editor.org/info/rfc4868>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", Version 2.0, May 2009, <http://www.secg.org/sec1-v2.pdf>.

[SHS] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

10.2. Informative References

[AEAD-CBC-SHA]
McGrew, D., Foley, J., and K. Paterson, "Authenticated Encryption with AES-CBC and HMAC-SHA", Work in Progress, draft-mcgrew-aead-aes-cbc-hmac-sha2-05, July 2014.

[CanvasApp]
Facebook, "Canvas Applications", 2010, <http://developers.facebook.com/docs/authentication/canvas>.

[JCA] Oracle, "Java Cryptography Architecture (JCA) Reference Guide", 2014, <http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>.

[JSE] Bradley, J. and N. Sakimura (editor), "JSON Simple Encryption", September 2010, <http://jsonenc.info/enc/1.0/>.

[JSMS] Rescorla, E. and J. Hildebrand, "JavaScript Message Security Format", Work in Progress, draft-rescorla-jsms-00, March 2011.

[JSS] Bradley, J. and N. Sakimura, Ed., "JSON Simple Sign 1.0", Draft 01, September 2010, <http://jsonenc.info/jss/1.0/>.

[JWE-JWK] Miller, M., "Using JavaScript Object Notation (JSON) Web Encryption (JWE) for Protecting JSON Web Key (JWK) Objects", Work in Progress, draft-miller-jose-jwe-protected-jwk-02, June 2013.

[MagicSignatures] Panzer, J., Ed., Laurie, B., and D. Balfanz, "Magic Signatures", January 2011, <http://salmon-protocol.googlecode.com/svn/trunk/draft-panzer-magicsig-01.html>.

[NIST.800-107] National Institute of Standards and Technology (NIST), "Recommendation for Applications Using Approved Hash Algorithms", NIST Special Publication 800-107, Revision 1, August 2012, <http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf>.

[NIST.800-63-2] National Institute of Standards and Technology (NIST), "Electronic Authentication Guideline", NIST Special Publication 800-63-2, August 2013, <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf>.

[PRECIS] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", Work in Progress, draft-ietf-precis-saslprepbis-16, April 2015.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <http://www.rfc-editor.org/info/rfc2631>.

[RFC3275] Eastlake 3rd, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, DOI 10.17487/RFC3275, March 2002, <http://www.rfc-editor.org/info/rfc3275>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <http://www.rfc-editor.org/info/rfc5116>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[W3C.NOTE-xmldsig-core2-20130411]
Eastlake, D., Reagle, J., Solo, D., Hirsch, F., Roessler, T., Yiu, K., Datta, P., and S. Cantor, "XML Signature Syntax and Processing Version 2.0", World Wide Web Consortium Note NOTE-xmldsig-core2-20130411, April 2013, <http://www.w3.org/TR/2013/NOTE-xmldsig-core2-20130411/>.

[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>.

[W3C.REC-xmlenc-core1-20130411]
Eastlake, D., Reagle, J., Hirsch, F., and T. Roessler, "XML Encryption Syntax and Processing Version 1.1", World Wide Web Consortium Recommendation REC-xmlenc-core1-20130411, April 2013, <http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/>.

Appendix A. Algorithm Identifier Cross-Reference

This appendix contains tables cross-referencing the cryptographic algorithm identifier values defined in this specification with the equivalent identifiers used by other standards and software packages. See XML DSIG [RFC3275], XML DSIG 2.0 [W3C.NOTE-xmldsig-core2-20130411], XML Encryption [W3C.REC-xmlenc-core-20021210], XML Encryption 1.1 [W3C.REC-xmlenc-core1-20130411], and Java Cryptography Architecture [JCA] for more information about the names defined by those documents.

A.1. Digital Signature/MAC Algorithm Identifier Cross-Reference

This section contains a table cross-referencing the JWS digital signature and MAC "alg" (algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

JWS	XML DSIG	
	JCA	OID
HS256	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256	
	HmacSHA256	1.2.840.113549.2.9
HS384	http://www.w3.org/2001/04/xmldsig-more#hmac-sha384	
	HmacSHA384	1.2.840.113549.2.10
HS512	http://www.w3.org/2001/04/xmldsig-more#hmac-sha512	
	HmacSHA512	1.2.840.113549.2.11
RS256	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256	
	SHA256withRSA	1.2.840.113549.1.1.11
RS384	http://www.w3.org/2001/04/xmldsig-more#rsa-sha384	
	SHA384withRSA	1.2.840.113549.1.1.12
RS512	http://www.w3.org/2001/04/xmldsig-more#rsa-sha512	
	SHA512withRSA	1.2.840.113549.1.1.13
ES256	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256	
	SHA256withECDSA	1.2.840.10045.4.3.2
ES384	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384	
	SHA384withECDSA	1.2.840.10045.4.3.3
ES512	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512	
	SHA512withECDSA	1.2.840.10045.4.3.4
PS256	http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1	
	SHA256withRSAandMGF1	1.2.840.113549.1.1.10
PS384	http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1	
	SHA384withRSAandMGF1	1.2.840.113549.1.1.10
PS512	http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1	
	SHA512withRSAandMGF1	1.2.840.113549.1.1.10

A.2. Key Management Algorithm Identifier Cross-Reference

This section contains a table cross-referencing the JWE "alg" (algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

JWE	XML ENC	
JCA		OID
RSA1_5	http://www.w3.org/2001/04/xmlenc#rsa-1_5	
RSA/ECB/PKCS1Padding		1.2.840.113549.1.1.1
RSA-OAEP	http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p	
RSA/ECB/OAEPWithSHA-1AndMGF1Padding		1.2.840.113549.1.1.7
RSA-OAEP-256	http://www.w3.org/2009/xmlenc11#rsa-oaep & http://www.w3.org/2009/xmlenc11#mgf1sha256	
RSA/ECB/OAEPWithSHA-256AndMGF1Padding & MGF1ParameterSpec.SHA256		1.2.840.113549.1.1.7
ECDH-ES	http://www.w3.org/2009/xmlenc11#ECDH-ES	
ECDH		1.3.132.1.12
A128KW	http://www.w3.org/2001/04/xmlenc#kw-aes128	
AESWrap		2.16.840.1.101.3.4.1.5
A192KW	http://www.w3.org/2001/04/xmlenc#kw-aes192	
AESWrap		2.16.840.1.101.3.4.1.25
A256KW	http://www.w3.org/2001/04/xmlenc#kw-aes256	
AESWrap		2.16.840.1.101.3.4.1.45

A.3. Content Encryption Algorithm Identifier Cross-Reference

This section contains a table cross-referencing the JWE "enc" (encryption algorithm) values defined in this specification with the equivalent identifiers used by other standards and software packages.

For the composite algorithms "A128CBC-HS256", "A192CBC-HS384", and "A256CBC-HS512", the corresponding AES-CBC algorithm identifiers are listed.

JWE	XML ENC	
JCA	OID	
A128CBC-HS256	http://www.w3.org/2001/04/xmlenc#aes128-cbc	
AES/CBC/PKCS5Padding	2.16.840.1.101.3.4.1.2	
A192CBC-HS384	http://www.w3.org/2001/04/xmlenc#aes192-cbc	
AES/CBC/PKCS5Padding	2.16.840.1.101.3.4.1.22	
A256CBC-HS512	http://www.w3.org/2001/04/xmlenc#aes256-cbc	
AES/CBC/PKCS5Padding	2.16.840.1.101.3.4.1.42	
A128GCM	http://www.w3.org/2009/xmlenc11#aes128-gcm	
AES/GCM/NoPadding	2.16.840.1.101.3.4.1.6	
A192GCM	http://www.w3.org/2009/xmlenc11#aes192-gcm	
AES/GCM/NoPadding	2.16.840.1.101.3.4.1.26	
A256GCM	http://www.w3.org/2009/xmlenc11#aes256-gcm	
AES/GCM/NoPadding	2.16.840.1.101.3.4.1.46	

Appendix B. Test Cases for AES_CBC_HMAC_SHA2 Algorithms

The following test cases can be used to validate implementations of the AES_CBC_HMAC_SHA2 algorithms defined in Section 5.2. They are also intended to correspond to test cases that may appear in a future version of [AEAD-CBC-SHA], demonstrating that the cryptographic computations performed are the same.

The variable names are those defined in Section 5.2. All values are hexadecimal.

B.1. Test Cases for AES_128_CBC_HMAC_SHA_256

AES_128_CBC_HMAC_SHA_256

```
K =       00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

ENC_KEY = 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

P =       41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
          6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
          69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
          74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
          65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
          6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
          20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
          75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =      1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =       54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
          69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
          4b 65 72 63 6b 68 6f 66 66 73

AL =      00 00 00 00 00 00 01 50

E =       c8 0e df a3 2d df 39 d5 ef 00 c0 b4 68 83 42 79
          a2 e4 6a 1b 80 49 f7 92 f7 6b fe 54 b9 03 a9 c9
          a9 4a c9 b4 7a d2 65 5c 5f 10 f9 ae f7 14 27 e2
          fc 6f 9b 3f 39 9a 22 14 89 f1 63 62 c7 03 23 36
          09 d4 5a c6 98 64 e3 32 1c f8 29 35 ac 40 96 c8
          6e 13 33 14 c5 40 19 e8 ca 79 80 df a4 b9 cf 1b
          38 4c 48 6f 3a 54 c5 10 78 15 8e e5 d7 9d e5 9f
          bd 34 d8 48 b3 d6 95 50 a6 76 46 34 44 27 ad e5
          4b 88 51 ff b5 98 f7 f8 00 74 b9 47 3c 82 e2 db

M =       65 2c 3f a3 6b 0a 7c 5b 32 19 fa b3 a3 0b c1 c4
          e6 e5 45 82 47 65 15 f0 ad 9f 75 a2 b7 1c 73 ef

T =       65 2c 3f a3 6b 0a 7c 5b 32 19 fa b3 a3 0b c1 c4
```

B.2. Test Cases for AES_192_CBC_HMAC_SHA_384

```
K =       00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
          20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17

ENC_KEY = 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27
          28 29 2a 2b 2c 2d 2e 2f

P =       41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
          6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
          69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
          74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
          65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
          6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
          20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
          75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =      1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =       54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
          69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
          4b 65 72 63 6b 68 6f 66 66 73

AL =      00 00 00 00 00 00 01 50

E =       ea 65 da 6b 59 e6 1e db 41 9b e6 2d 19 71 2a e5
          d3 03 ee b5 00 52 d0 df d6 69 7f 77 22 4c 8e db
          00 0d 27 9b dc 14 c1 07 26 54 bd 30 94 42 30 c6
          57 be d4 ca 0c 9f 4a 84 66 f2 2b 22 6d 17 46 21
          4b f8 cf c2 40 0a dd 9f 51 26 e4 79 66 3f c9 0b
          3b ed 78 7a 2f 0f fc bf 39 04 be 2a 64 1d 5c 21
          05 bf e5 91 ba e2 3b 1d 74 49 e5 32 ee f6 0a 9a
          c8 bb 6c 6b 01 d3 5d 49 78 7b cd 57 ef 48 49 27
          f2 80 ad c9 1a c0 c4 e7 9c 7b 11 ef c6 00 54 e3

M =       84 90 ac 0e 58 94 9b fe 51 87 5d 73 3f 93 ac 20
          75 16 80 39 cc c7 33 d7 45 94 f8 86 b3 fa af d4
          86 f2 5c 71 31 e3 28 1e 36 c7 a2 d1 30 af de 57

T =       84 90 ac 0e 58 94 9b fe 51 87 5d 73 3f 93 ac 20
          75 16 80 39 cc c7 33 d7
```

B.3. Test Cases for AES_256_CBC_HMAC_SHA_512

```
K =       00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
          20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
          30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

ENC_KEY = 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
          30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f

P =       41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
          6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
          69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
          74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
          65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
          6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
          20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
          75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =      1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =       54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
          69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
          4b 65 72 63 6b 68 6f 66 66 73

AL =      00 00 00 00 00 00 01 50

E =       4a ff aa ad b7 8c 31 c5 da 4b 1b 59 0d 10 ff bd
          3d d8 d5 d3 02 42 35 26 91 2d a0 37 ec bc c7 bd
          82 2c 30 1d d6 7c 37 3b cc b5 84 ad 3e 92 79 c2
          e6 d1 2a 13 74 b7 7f 07 75 53 df 82 94 10 44 6b
          36 eb d9 70 66 29 6a e6 42 7e a7 5c 2e 08 46 a1
          1a 09 cc f5 37 0d c8 0b fe cb ad 28 c7 3f 09 b3
          a3 b7 5e 66 2a 25 94 41 0a e4 96 b2 e2 e6 60 9e
          31 e6 e0 2c c8 37 f0 53 d2 1f 37 ff 4f 51 95 0b
          be 26 38 d0 9d d7 a4 93 09 30 80 6d 07 03 b1 f6

M =       4d d3 b4 c0 88 a7 f4 5c 21 68 39 64 5b 20 12 bf
          2e 62 69 a8 c5 6a 81 6d bc 1b 26 77 61 95 5b c5
          fd 30 a5 65 c6 16 ff b2 f3 64 ba ec e6 8f c4 07
          53 bc fc 02 5d de 36 93 75 4a a1 f5 c3 37 3b 9c

T =       4d d3 b4 c0 88 a7 f4 5c 21 68 39 64 5b 20 12 bf
          2e 62 69 a8 c5 6a 81 6d bc 1b 26 77 61 95 5b c5
```

Appendix C. Example ECDH-ES Key Agreement Computation

This example uses ECDH-ES Key Agreement and the Concat KDF to derive the CEK in the manner described in Section 4.6. In this example, the ECDH-ES Direct Key Agreement mode ("alg" value "ECDH-ES") is used to produce an agreed-upon key for AES GCM with a 128-bit key ("enc" value "A128GCM").

In this example, a producer Alice is encrypting content to a consumer Bob. The producer (Alice) generates an ephemeral key for the key agreement computation. Alice's ephemeral key (in JWK format) used for the key agreement computation in this example (including the private part) is:

```
{"kty":"EC",
 "crv":"P-256",
 "x":"gI0GAILBdu7T53akrFmMyGcsF3n5dO7MmwNBHKW5SV0",
 "y":"SLW_xSffzlPWrHEVI30DHM_4egVwt3NQqeUD7nMFpps",
 "d":"0_NxaRPUMQoAJt50Gz8YiTr8gRTwyEaCumd-MToTmIo"
}
```

The consumer's (Bob's) key (in JWK format) used for the key agreement computation in this example (including the private part) is:

```
{"kty":"EC",
 "crv":"P-256",
 "x":"weNJy2HscCSM6AEDTDg04biOvhFhyyWvOHQfeF_PxMQ",
 "y":"e8lnCO-AlStT-NJVX-crhB7QRYhiix03illJOVAOyck",
 "d":"VEmDZpDXXK8p8N0Cndsxs924q6nS1RXFASRl6BfUqdw"
}
```

Header Parameter values used in this example are as follows. The "apu" (agreement PartyUInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Alice" and the "apv" (agreement PartyVInfo) Header Parameter value is the base64url encoding of the UTF-8 string "Bob". The "epk" (ephemeral public key) Header Parameter is used to communicate the producer's (Alice's) ephemeral public key value to the consumer (Bob).

```
  {"alg":"ECDH-ES",
   "enc":"A128GCM",
   "apu":"QWxpY2U",
   "apv":"Qm9i",
   "epk":
    {"kty":"EC",
     "crv":"P-256",
     "x":"gI0GAILBdu7T53akrFmMyGcsF3n5dO7MmwNBHKW5SV0",
     "y":"SLW_xSffzlPWrHEVI30DHM_4egVwt3NQqeUD7nMFpps"
    }
  }
```

The resulting Concat KDF [NIST.800-56A] parameter values are:

Z
: This is set to the ECDH-ES key agreement output. (This value is often not directly exposed by libraries, due to NIST security requirements, and only serves as an input to a KDF.) In this example, Z is following the octet sequence (using JSON array notation):
[158, 86, 217, 29, 129, 113, 53, 211, 114, 131, 66, 131, 191, 132, 38, 156, 251, 49, 110, 163, 218, 128, 106, 72, 246, 218, 167, 121, 140, 254, 144, 196].

keydatalen
: This value is 128 - the number of bits in the desired output key (because "A128GCM" uses a 128-bit key).

AlgorithmID
: This is set to the octets representing the 32-bit big-endian value 7 - [0, 0, 0, 7] - the number of octets in the AlgorithmID content "A128GCM", followed, by the octets representing the ASCII string "A128GCM" - [65, 49, 50, 56, 71, 67, 77].

PartyUInfo
: This is set to the octets representing the 32-bit big-endian value 5 - [0, 0, 0, 5] - the number of octets in the PartyUInfo content "Alice", followed, by the octets representing the UTF-8 string "Alice" - [65, 108, 105, 99, 101].

PartyVInfo
: This is set to the octets representing the 32-bit big-endian value 3 - [0, 0, 0, 3] - the number of octets in the PartyUInfo content "Bob", followed, by the octets representing the UTF-8 string "Bob" - [66, 111, 98].

SuppPubInfo
This is set to the octets representing the 32-bit big-endian value 128 - [0, 0, 0, 128] - the keydatalen value.

SuppPrivInfo
This is set to the empty octet sequence.

Concatenating the parameters AlgorithmID through SuppPubInfo results in an OtherInfo value of:
[0, 0, 0, 7, 65, 49, 50, 56, 71, 67, 77, 0, 0, 0, 5, 65, 108, 105, 99, 101, 0, 0, 0, 3, 66, 111, 98, 0, 0, 0, 128]

Concatenating the round number 1 ([0, 0, 0, 1]), Z, and the OtherInfo value results in the Concat KDF round 1 hash input of:
[0, 0, 0, 1,
158, 86, 217, 29, 129, 113, 53, 211, 114, 131, 66, 131, 191, 132, 38, 156, 251, 49, 110, 163, 218, 128, 106, 72, 246, 218, 167, 121, 140, 254, 144, 196,
0, 0, 0, 7, 65, 49, 50, 56, 71, 67, 77, 0, 0, 0, 5, 65, 108, 105, 99, 101, 0, 0, 0, 3, 66, 111, 98, 0, 0, 0, 128]

The resulting derived key, which is the first 128 bits of the round 1 hash output is:
[86, 170, 141, 234, 248, 35, 109, 32, 92, 34, 40, 205, 113, 167, 16, 26]

The base64url-encoded representation of this derived key is:

VqqN6vgjbSBcIijNcacQGg

Acknowledgements

Solutions for signing and encrypting JSON content were previously explored by "Magic Signatures" [MagicSignatures], "JSON Simple Sign 1.0" [JSS], "Canvas Applications" [CanvasApp], "JSON Simple Encryption" [JSE], and "JavaScript Message Security Format" [JSMS], all of which influenced this document.

The "Authenticated Encryption with AES-CBC and HMAC-SHA" [AEAD-CBC-SHA] specification, upon which the AES_CBC_HMAC_SHA2 algorithms are based, was written by David A. McGrew and Kenny Paterson. The test cases for AES_CBC_HMAC_SHA2 are based upon those for [AEAD-CBC-SHA] by John Foley.

Matt Miller wrote "Using JavaScript Object Notation (JSON) Web Encryption (JWE) for Protecting JSON Web Key (JWK) Objects" [JWE-JWK], upon which the password-based encryption content of this document is based.

This specification is the work of the JOSE working group, which includes dozens of active and dedicated participants. In particular, the following individuals contributed ideas, feedback, and wording that influenced this specification:

Dirk Balfanz, Richard Barnes, Carsten Bormann, John Bradley, Brian Campbell, Alissa Cooper, Breno de Medeiros, Vladimir Dzhuvinov, Roni Even, Stephen Farrell, Yaron Y. Goland, Dick Hardt, Joe Hildebrand, Jeff Hodges, Edmund Jay, Charlie Kaufman, Barry Leiba, James Manger, Matt Miller, Kathleen Moriarty, Tony Nadalin, Axel Nennker, John Panzer, Emmanuel Raviart, Eric Rescorla, Pete Resnick, Nat Sakimura, Jim Schaad, Hannes Tschofenig, and Sean Turner.

Jim Schaad and Karen O'Donoghue chaired the JOSE working group and Sean Turner, Stephen Farrell, and Kathleen Moriarty served as Security Area Directors during the creation of this specification.

Author's Address

Michael B. Jones
Microsoft

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