Abstract

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a JSON Web Key Set (JWK Set) JSON data structure that represents a set of JWKs. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries defined by that specification.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 27, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
Table of Contents

1.  Introduction ........................................... 4
   1.1.  Notational Conventions .......................... 4

2.  Terminology ............................................ 4

3.  Example JWK ............................................ 5

4.  JSON Web Key (JWK) Format ............................. 5
   4.1.  "kty" (Key Type) Parameter ....................... 6
   4.2.  "use" (Public Key Use) Parameter .................. 6
   4.3.  "key_ops" (Key Operations) Parameter .............. 7
   4.4.  "alg" (Algorithm) Parameter ....................... 8
   4.5.  "kid" (Key ID) Parameter .......................... 8
   4.6.  "x5u" (X.509 URL) Parameter ....................... 8
   4.7.  "x5c" (X.509 Certificate Chain) Parameter .......... 9
   4.8.  "x5t" (X.509 Certificate SHA-1 Thumbprint) Parameter ... 9
   4.9.  "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) Parameter .... 10

5.  JSON Web Key Set (JWK Set) Format .................... 10
   5.1.  "keys" Parameter .................................. 10

6.  String Comparison Rules ............................... 11

7.  Encrypted JWK and Encrypted JWK Set Formats .......... 11

8.  IANA Considerations .................................. 11
   8.1.  JSON Web Key Parameters Registry .................. 13
         8.1.1.  Registration Template ......................... 13
         8.1.2.  Initial Registry Contents ..................... 14
   8.2.  JSON Web Key Use Registry ........................ 15
         8.2.1.  Registration Template ......................... 15
         8.2.2.  Initial Registry Contents ..................... 16
   8.3.  JSON Web Key Operations Registry ................ 16
         8.3.1.  Registration Template ......................... 16
         8.3.2.  Initial Registry Contents ..................... 17
   8.4.  JSON Web Key Set Parameters Registry .............. 18
         8.4.1.  Registration Template ......................... 18
         8.4.2.  Initial Registry Contents ..................... 19
   8.5.  Media Type Registration ........................... 19
         8.5.1.  Registry Contents .............................. 19

9.  Security Considerations ................................ 20
   9.1.  Key Provenance and Trust ......................... 20
   9.2.  Preventing Disclosure of Non-Public Key Information .. 20
   9.3.  RSA Private Key Representations and Blinding ....... 21
   9.4.  Key Entropy and Random Values .................... 21

10. References ........................................... 21
   10.1.  Normative References ............................ 21
   10.2.  Informative References ........................... 23
1. Introduction

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) [RFC7159] data structure that represents a cryptographic key. This specification also defines a JSON Web Key Set (JWK Set) JSON data structure that represents a set of JWKs. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) [JWA] specification and IANA registries defined by that specification.

Goals for this specification do not include representing new kinds of certificate chains, representing new kinds of certified keys, or replacing X.509 certificates.

JWKs and JWK Sets are used in the JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] specifications.

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]. If these words are used without being spelled in uppercase then they are to be interpreted with their normal natural language meanings.

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.

ASCII(STRING) denotes the octets of the ASCII [RFC20] representation of STRING.

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

2. Terminology

These terms defined by the JSON Web Signature (JWS) [JWS] specification are incorporated into this specification: "Base64url Encoding", "Collision-Resistant Name", "Header Parameter", and "JOSE Header".
These terms defined by the Internet Security Glossary, Version 2 [RFC4949] are incorporated into this specification: "Ciphertext", "Digital Signature", "Message Authentication Code (MAC)", and "Plaintext".

These terms are defined by this specification:

JSON Web Key (JWK)
   A JSON object that represents a cryptographic key. The members of the object represent properties of the key, including its value.

JSON Web Key Set (JWK Set)
   A JSON object that represents a set of JWKs. The JSON object MUST have a "keys" member, which is an array of JWK objects.

3. Example JWK

This section provides an example of a JWK. The following example JWK declares that the key is an Elliptic Curve [DSS] key, it is used with the P-256 Elliptic Curve, and its x and y coordinates are the base64url encoded values shown. A key identifier is also provided for the key.

```
{"kty":"EC",
 "crv":"P-256",
 "x":"f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
 "y":"x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0",
 "kid":"Public key used in JWS A.3 example"
}
```

Additional example JWK values can be found in Appendix A.

4. JSON Web Key (JWK) Format

A JSON Web Key (JWK) is a JSON object that represents a cryptographic key. The members of the object represent properties of the key, including its value. This JSON object MAY contain white space and/or line breaks before or after any JSON values or structural characters, in accordance with Section 2 of RFC 7159 [RFC7159]. This document defines the key parameters that are not algorithm specific, and thus common to many keys.

In addition to the common parameters, each JWK will have members that are key type-specific. These members represent the parameters of the key. Section 6 of the JSON Web Algorithms (JWA) [JWA] specification defines multiple kinds of cryptographic keys and their associated
members.

The member names within a JWK MUST be unique; JWK parsers MUST either reject JWKs with duplicate member names or use a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 (The JSON Object) of ECMAScript 5.1 [ECMAScript].

Additional members can be present in the JWK; if not understood by implementations encountering them, they MUST be ignored. Member names used for representing key parameters for different keys types need not be distinct. Any new member name should either be registered in the IANA JSON Web Key Parameters registry defined in Section 8.1 or be a value that contains a Collision-Resistant Name.

4.1. "kty" (Key Type) Parameter

The "kty" (key type) member identifies the cryptographic algorithm family used with the key, such as "RSA" or "EC". "kty" values should either be registered in the IANA JSON Web Key Types registry defined in [JWA] or be a value that contains a Collision-Resistant Name. The "kty" value is a case-sensitive string. This member MUST be present in a JWK.

A list of defined "kty" values can be found in the IANA JSON Web Key Types registry defined in [JWA]; the initial contents of this registry are the values defined in Section 6.1 of the JSON Web Algorithms (JWA) [JWA] specification.

The key type definitions include specification of the members to be used for those key types. Additional members used with "kty" values can also be found in the IANA JSON Web Key Parameters registry defined in Section 8.1.

4.2. "use" (Public Key Use) Parameter

The "use" (public key use) member identifies the intended use of the public key. The "use" parameter is employed to indicate whether a public key is used for encrypting data or verifying the signature on data.

Values defined by this specification are:

- "sig" (signature)
- "enc" (encryption)

Other values MAY be used. The "use" value is a case-sensitive string. Use of the "use" member is OPTIONAL, unless the application requires its presence.
When a key is used to wrap another key and a Public Key Use designation for the first key is desired, the "enc" (encryption) key use value is used, since key wrapping is a kind of encryption. The "enc" value is also be used for public keys used for key agreement operations.

Additional Public Key Use values can be registered in the IANA JSON Web Key Use registry defined in Section 8.2. Registering any extension values used is highly recommended when this specification is used in open environments, in which multiple organizations need to have a common understanding of any extensions used. However, unregistered extension values can be used in closed environments, in which the producing and consuming organization will always be the same.

4.3. "key_ops" (Key Operations) Parameter

The "key_ops" (key operations) member identifies the operation(s) that the key is intended to be used for. The "key_ops" parameter is intended for use cases in which public, private, or symmetric keys may be present.

Its value is an array of key operation values. Values defined by this specification are:

- "sign" (compute digital signature or MAC)
- "verify" (verify digital signature or MAC)
- "encrypt" (encrypt content)
- "decrypt" (decrypt content and validate decryption, if applicable)
- "wrapKey" (encrypt key)
- "unwrapKey" (decrypt key and validate decryption, if applicable)
- "deriveKey" (derive key)
- "deriveBits" (derive bits not to be used as a key)

(Note that the "key_ops" values intentionally match the "KeyUsage" values defined in the Web Cryptography API [WebCrypto] specification.)

Other values MAY be used. The key operation values are case-sensitive strings. Duplicate key operation values MUST NOT be present in the array. Use of the "key_ops" member is OPTIONAL, unless the application requires its presence.

Multiple unrelated key operations SHOULD NOT be specified for a key because of the potential vulnerabilities associated with using the same key with multiple algorithms. Thus, the combinations "sign" with "verify", "encrypt" with "decrypt", and "wrapKey" with "unwrapKey" are permitted, but other combinations SHOULD NOT be used.
Additional Key Operations values can be registered in the IANA JSON Web Key Operations registry defined in Section 8.3. The same considerations about registering extension values apply to the "key_ops" member as do for the "use" member.

The "use" and "key_ops" JWK members SHOULD NOT be used together; however, if both are used, the information they convey MUST be consistent. Applications should specify which of these members they use, if either is to be used by the application.

4.4. "alg" (Algorithm) Parameter

The "alg" (algorithm) member identifies the algorithm intended for use with the key. The values used should either be registered in the IANA JSON Web Signature and Encryption Algorithms registry defined in [JWA] or be a value that contains a Collision-Resistant Name. Use of this member is OPTIONAL.

4.5. "kid" (Key ID) Parameter

The "kid" (key ID) member is used to match a specific key. This is used, for instance, to choose among a set of keys within a JWK Set during key rollover. The structure of the "kid" value is unspecified. When "kid" values are used within a JWK Set, different keys within the JWK Set SHOULD use distinct "kid" values. (One example in which different keys might use the same "kid" value is if they have different "kty" (key type) values but are considered to be equivalent alternatives by the application using them.) The "kid" value is a case-sensitive string. Use of this member is OPTIONAL.

When used with JWS or JWE, the "kid" value is used to match a JWS or JWE "kid" Header Parameter value.

4.6. "x5u" (X.509 URL) Parameter

The "x5u" (X.509 URL) member is a URI [RFC3986] that refers to a resource for an X.509 public key certificate or certificate chain [RFC5280]. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to RFC 5280 [RFC5280] in PEM encoded form, with each certificate delimited as specified in Section 6.1 of RFC 4945 [RFC4945]. The key in the first certificate MUST match the public key represented by other members of the JWK. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS [RFC2818, RFC5246]; the identity of the server MUST be validated, as per Section 6 of RFC 6125 [RFC6125]. Use of this member is OPTIONAL.
While there is no requirement that optional JWK members providing key usage, algorithm, or other information be present when the "x5u" member is used, doing so may improve interoperability for applications that do not handle PKIX certificates. If other members are present, the contents of those members MUST be semantically consistent with the related fields in the first certificate. For instance, if the "use" member is present, then it MUST correspond to the usage that is specified in the certificate, when it includes this information. Similarly, if the "alg" member is present, it MUST correspond to the algorithm specified in the certificate.

4.7. "x5c" (X.509 Certificate Chain) Parameter

The "x5c" (X.509 Certificate Chain) member contains a chain of one or more PKIX certificates [RFC5280]. The certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64 encoded ([RFC4648] Section 4 -- not base64url encoded) DER [ITU.X690.1994] PKIX certificate value. The PKIX certificate containing the key value MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The key in the first certificate MUST match the public key represented by other members of the JWK. Use of this member is OPTIONAL.

As with the "x5u" member, optional JWK members providing key usage, algorithm, or other information MAY also be present when the "x5c" member is used. If other members are present, the contents of those members MUST be semantically consistent with the related fields in the first certificate. See the last paragraph of Section 4.6 for additional guidance on this.

4.8. "x5t" (X.509 Certificate SHA-1 Thumbprint) Parameter

The "x5t" (X.509 Certificate SHA-1 Thumbprint) member is a base64url encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. Note that certificate thumbprints are also sometimes known as certificate fingerprints. The key in the certificate MUST match the public key represented by other members of the JWK. Use of this member is OPTIONAL.

As with the "x5u" member, optional JWK members providing key usage, algorithm, or other information MAY also be present when the "x5t" member is used. If other members are present, the contents of those members MUST be semantically consistent with the related fields in the referenced certificate. See the last paragraph of Section 4.6 for additional guidance on this.
4.9. "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) Parameter

The "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) member is a base64url encoded SHA-256 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. Note that certificate thumbprints are also sometimes known as certificate fingerprints. The key in the certificate MUST match the public key represented by other members of the JWK. Use of this member is OPTIONAL.

As with the "x5u" member, optional JWK members providing key usage, algorithm, or other information MAY also be present when the "x5t#S256" member is used. If other members are present, the contents of those members MUST be semantically consistent with the related fields in the referenced certificate. See the last paragraph of Section 4.6 for additional guidance on this.

5. JSON Web Key Set (JWK Set) Format

A JSON Web Key Set (JWK Set) is a JSON object that represents a set of JWKs. The JSON object MUST have a "keys" member, with its value being an array of JWK objects. This JSON object MAY contain white space and/or line breaks.

The member names within a JWK Set MUST be unique; JWK Set parsers MUST either reject JWK Sets with duplicate member names or use a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 (The JSON Object) of ECMAScript 5.1 [ECMAScript].

Additional members can be present in the JWK Set; if not understood by implementations encountering them, they MUST be ignored. Parameters for representing additional properties of JWK Sets should either be registered in the IANA JSON Web Key Set Parameters registry defined in Section 8.4 or be a value that contains a Collision-Resistant Name.

Implementations SHOULD ignore JWKs within a JWK Set that use "kty" (key type) values that are not understood by them, are missing required members, or for which values are out of the supported ranges.

5.1. "keys" Parameter

The value of the "keys" member is an array of JWK values. By default, the order of the JWK values within the array does not imply an order of preference among them, although applications of JWK Sets can choose to assign a meaning to the order for their purposes, if
desired.

6. String Comparison Rules

The string comparison rules for this specification are the same as those defined in Section 5.3 of [JWS].

7. Encrypted JWK and Encrypted JWK Set Formats

Access to JWKs containing non-public key material by parties without legitimate access to the non-public information MUST be prevented. This can be accomplished by encrypting the JWK when potentially observable by such parties to prevent the disclosure of private or symmetric key values. The use of an Encrypted JWK, which is a JWE with the UTF-8 encoding of a JWK as its plaintext value, is recommended for this purpose. The processing of Encrypted JWKs is identical to the processing of other JWEs. A "cty" (content type) Header Parameter value of "jwk+json" MUST be used to indicate that the content of the JWE is a JWK, unless the application knows that the encrypted content is a JWK by another means or convention, in which case the "cty" value would typically be omitted.

JWK Sets containing non-public key material will also need to be encrypted under these circumstances. The use of an Encrypted JWK Set, which is a JWE with the UTF-8 encoding of a JWK Set as its plaintext value, is recommended for this purpose. The processing of Encrypted JWK Sets is identical to the processing of other JWEs. A "cty" (content type) Header Parameter value of "jwk-set+json" MUST be used to indicate that the content of the JWE is a JWK Set, unless the application knows that the encrypted content is a JWK Set by another means or convention, in which case the "cty" value would typically be omitted.

See Appendix C for an example encrypted JWK.

8. IANA Considerations

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

Values are registered on a Specification Required [RFC5226] basis 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 Expert(s) may approve registration once they are
satisfied that such a specification will be published.

Registration requests must be sent to the jose-reg-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for access token type: example").

Within the review period, the Designated Expert(s) 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 Expert(s) includes determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration description is clear.

IANA must only accept registry updates from the Designated Expert(s) 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 Expert(s).

[[ Note to the RFC Editor and IANA: Pearl Liang of ICANN had requested that the draft supply the following proposed registry description information. It is to be used for all registries established by this specification.

- Protocol Category: JSON Object Signing and Encryption (JOSE)
- Registry Location: http://www.iana.org/assignments/jose
- Webpage Title: (same as the protocol category)
- Registry Name: (same as the section title, but excluding the word "Registry", for example "JSON Web Key Parameters")
]]
8.1. JSON Web Key Parameters Registry

This specification establishes the IANA JSON Web Key Parameters registry for JWK parameter names. The registry records the parameter name, the key type(s) that the parameter is used with, and a reference to the specification that defines it. It also records whether the parameter conveys public or private information. This specification registers the parameter names defined in Section 4.

The same JWK parameter name may be registered multiple times, provided that duplicate parameter registrations are only for key type specific JWK parameters; in this case, the meaning of the duplicate parameter name is disambiguated by the "kty" value of the JWK containing it.

8.1.1. Registration Template

Parameter Name:
The name requested (e.g., "kid"). 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 Expert(s) state that there is a compelling reason to allow an exception in this particular case. However, matching names may be registered, provided that the accompanying sets of "kty" values that the Parameter Name is used with are disjoint; for the purposes of matching "kty" values, "*" matches all values.

Parameter Description:
Brief description of the parameter (e.g., "Key ID").

Used with "kty" Value(s):
The key type parameter value(s) that the parameter name is to be used with, or the value "*" if the parameter value is used with all key types. Values may not match other registered "kty" values in a case-insensitive manner when the registered Parameter Name is the same (including when the Parameter Name matches in a case-insensitive manner) unless the Designated Expert(s) state that there is a compelling reason to allow an exception in this particular case.

Parameter Information Class:
Registers whether the parameter conveys public or private information. Its value must be one the words Public or Private.
8.1.2. Initial Registry Contents

- Parameter Name: "kty"
  - Parameter Description: Key Type
  - Used with "kty" Value(s): *
  - Parameter Information Class: Public
  - Change Controller: IESG
  - Specification Document(s): Section 4.1 of [[ this document ]]

- Parameter Name: "use"
  - Parameter Description: Public Key Use
  - Used with "kty" Value(s): *
  - Parameter Information Class: Public
  - Change Controller: IESG
  - Specification Document(s): Section 4.2 of [[ this document ]]

- Parameter Name: "key_ops"
  - Parameter Description: Key Operations
  - Used with "kty" Value(s): *
  - Parameter Information Class: Public
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Parameter Name: "alg"
  - Parameter Description: Algorithm
  - Used with "kty" Value(s): *
  - Parameter Information Class: Public
  - Change Controller: IESG
  - Specification Document(s): Section 4.4 of [[ this document ]]

- Parameter Name: "kid"
  - Parameter Description: Key ID
  - Used with "kty" Value(s): *
  - Parameter Information Class: Public
  - Change Controller: IESG
8.2. JSON Web Key Use Registry

This specification establishes the IANA JSON Web Key Use registry for JWK "use" (public key use) member values. The registry records the public key use value and a reference to the specification that defines it. This specification registers the parameter names defined in Section 4.2.

8.2.1. Registration Template

Use Member Value:

The name requested (e.g., "sig"). 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 Expert(s) state that there is a compelling reason to allow an exception in this
particular case.

Use Description:
Brief description of the use (e.g., "Digital Signature or MAC").

Change Controller:
For Standards Track RFCs, state "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(s) that specify the parameter, preferably including URI(s) that can be used to retrieve copies of the document(s). An indication of the relevant sections may also be included but is not required.

8.2.2. Initial Registry Contents

- Use Member Value: "sig"
  - Use Description: Digital Signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 4.2 of [[ this document ]]

- Use Member Value: "enc"
  - Use Description: Encryption
  - Change Controller: IESG
  - Specification Document(s): Section 4.2 of [[ this document ]]

8.3. JSON Web Key Operations Registry

This specification establishes the IANA JSON Web Key Operations registry for values of JWK "key_ops" array elements. The registry records the key operation value and a reference to the specification that defines it. This specification registers the parameter names defined in Section 4.3.

8.3.1. Registration Template

Key Operation Value:
The name requested (e.g., "sign"). 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 Expert(s) state that there is a compelling reason to allow an exception in this particular case.
Key Operation Description:
Brief description of the key operation (e.g., "Compute digital signature or MAC").

Change Controller:
For Standards Track RFCs, state "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(s) that specify the parameter, preferably including URI(s) that can be used to retrieve copies of the document(s). An indication of the relevant sections may also be included but is not required.

8.3.2. Initial Registry Contents

- Key Operation Value: "sign"
  - Key Operation Description: Compute digital signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Key Operation Value: "verify"
  - Key Operation Description: Verify digital signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Key Operation Value: "encrypt"
  - Key Operation Description: Encrypt content
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Key Operation Value: "decrypt"
  - Key Operation Description: Decrypt content and validate decryption, if applicable
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Key Operation Value: "wrapKey"
  - Key Operation Description: Encrypt key
  - Change Controller: IESG
  - Specification Document(s): Section 4.3 of [[ this document ]]

- Key Operation Value: "unwrapKey"
  - Key Operation Description: Decrypt key and validate decryption, if applicable
8.4. JSON Web Key Set Parameters Registry

This specification establishes the IANA JSON Web Key Set Parameters registry for JWK Set parameter names. The registry records the parameter name and a reference to the specification that defines it. This specification registers the parameter names defined in Section 5.

8.4.1. Registration Template

Parameter Name:
The name requested (e.g., "keys"). 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 Expert(s) state that there is a compelling reason to allow an exception in this particular case.

Parameter Description:
Brief description of the parameter (e.g., "Array of JWK values").

Change Controller:
For Standards Track RFCs, state "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(s) that specify the parameter, preferably including URI(s) that can be used to retrieve copies of the document(s). An indication of the relevant sections may also be included but is not required.
8.4.2. Initial Registry Contents

- Parameter Name: "keys"
- Parameter Description: Array of JWK values
- Change Controller: IESG
- Specification Document(s): Section 5.1 of [[ this document ]]

8.5. Media Type Registration

8.5.1. Registry Contents

This specification registers the "application/jwk+json" and "application/jwk-set+json" Media Types [RFC2046] in the MIME Media Types registry [IANA.MediaTypes] in the manner described in RFC 6838 [RFC6838], which can be used to indicate, respectively, that the content is a JWK or a JWK Set.

- Type Name: application
- Subtype Name: jwk+json
- Required Parameters: n/a
- Optional Parameters: n/a
- Encoding considerations: 8bit; application/jwk+json values are represented as a JSON object; UTF-8 encoding SHOULD be employed for the JSON object.
- Security Considerations: See the Security Considerations section of [[ this document ]]
- Interoperability Considerations: n/a
- Published Specification: [[ this document ]]
- Applications that use this media type: OpenID Connect, Salesforce, Google, Android, Windows Azure, W3C WebCrypto API, numerous others
- Fragment identifier considerations: n/a
- Additional Information: Magic number(s): n/a, File extension(s): n/a, Macintosh file type code(s): n/a
- Person & email address to contact for further information: Michael B. Jones, mbj@microsoft.com
- Intended Usage: COMMON
- Restrictions on Usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change Controller: IESG
- Provisional registration? No

- Type Name: application
- Subtype Name: jwk-set+json
- Required Parameters: n/a
- Optional Parameters: n/a
- Encoding considerations: 8bit; application/jwk-set+json values are represented as a JSON Object; UTF-8 encoding SHOULD be employed for the JSON object.
9. 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.

9.1. Key Provenance and Trust

One should place no more trust in the data cryptographically secured by a key than in the method by which it was obtained and in the trustworthiness of the entity asserting an association with the key. Any data associated with a key that is obtained in an untrusted manner should be treated with skepticism. See Section 10.3 of [JWS] for security considerations on key origin authentication.

The security considerations in Section 12.3 of XML DSIG 2.0 [W3C.NOTE-xmldsig-core2-20130411] about the strength of a digital signature depending upon all the links in the security chain also apply to this specification.

The TLS Requirements in Section 8 of [JWS] also apply to this specification.

9.2. Preventing Disclosure of Non-Public Key Information

Private and symmetric keys MUST be protected from disclosure to unintended parties. One recommended means of doing so is to encrypt JWks or JWK Sets containing them by using the JWK or JWK Set value as the plaintext of a JWE. Of course, this requires that there be a
secure way to obtain the key used to encrypt the non-public key information to the intended party and a secure way for that party to obtain the corresponding decryption key.

The security considerations in RFC 3447 [RFC3447] and RFC 6030 [RFC6030] about protecting private and symmetric keys, key usage, and information leakage also apply to this specification.

9.3. RSA Private Key Representations and Blinding

The RSA Key blinding operation [Kocher], which is a defense against some timing attacks, requires all of the RSA key values "n", "e", and "d". However, some RSA private key representations do not include the public exponent "e", but only include the modulus "n" and the private exponent "d". This is true, for instance, of the Java RSAPrivateKeySpec API, which does not include the public exponent "e" as a parameter. So as to enable RSA key blinding, such representations should be avoided. For Java, the RSAPrivateCrtKeySpec API can be used instead. Section 8.2.2(i) of the Handbook of Applied Cryptography [HAC] discusses how to compute the remaining RSA private key parameters, if needed, using only "n", "e", and "d".

9.4. Key Entropy and Random Values

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

10. References

10.1. Normative References

[ECMAScript]

[IANA.MediaTypes]
Internet Assigned Numbers Authority (IANA), "MIME Media Types", 2005.

[ITU.X690.1994]
Internet-Draft  JWK  October 2014

[JWA]  Jones, M., "JSON Web Algorithms (JWA)",
draft-ietf-jose-json-web-algorithms (work in progress),
October 2014.

draft-ietf-jose-json-web-encryption (work in progress),
October 2014.

[JWS]  Jones, M., Bradley, J., and N. Sakimura, "JSON Web
Signature (JWS)", draft-ietf-jose-json-web-signature (work
in progress), October 2014.

[RFC20] Cerf, V., "ASCII format for Network Interchange",
RFC 20, October 1969.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part Two: Media Types",
RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate


[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO

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

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

[RFC4945] Korver, B., "The Internet IP Security PKI Profile of
IKEv1/ISAKMP, IKEv2, and PKIX", RFC 4945, August 2007.

RFC 4949, August 2007.


[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
Housley, R., and W. Polk, "Internet X.509 Public Key
Infrastructure Certificate and Certificate Revocation List


10.2. Informative References


Appendix A.  Example JSON Web Key Sets

A.1.  Example Public Keys

The following example JWK Set contains two public keys represented as JWKs: one using an Elliptic Curve algorithm and a second one using an RSA algorithm.  The first specifies that the key is to be used for encryption.  The second specifies that the key is to be used with the "RS256" algorithm.  Both provide a Key ID for key matching purposes.  In both cases, integers are represented using the base64url encoding of their big endian representations.  (Long lines are broken are for display purposes only.)

```
{"keys":
 [
  {"kty":"EC",
   "crv":"P-256",
   "x":"MKBCTNiCUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
   "y":"4Etl6SRW2YiLURn5vfVHuhp7x8PxlmtWwlbbM4IFyM",
   "use":"enc",
   "kid":"1"},

  {"kty":"RSA",
   "n": "0vx7agoebGcQSuuPiLJXZptN9ndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCIvF4n3oknhMstn64tZ_2W-5JsGy4Hc5n9yBXArw1931qt7_RN5w6Cf0h4QyQ5v-65YGjQRO_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHu6qMQvRl5hajrn1n91CbOpbISD08qkNyrdkt-bFThhAI4vMQFh6weZu0fM4lf2NcRwr3XPksINHaQ-G_xBniIqbw0SljF44-csFCur-kEgUB8awapJzKnnqDKgw",
   "e":"AQAB",
   "alg":"RS256",
   "kid":"2011-04-29"}
  ]
}
```

A.2.  Example Private Keys

The following example JWK Set contains two keys represented as JWKs containing both public and private key values: one using an Elliptic Curve algorithm and a second one using an RSA algorithm.  This
example extends the example in the previous section, adding private key values. (Line breaks are for display purposes only.)

```json
{"keys": [
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "MKBCNICKUSDIll1ySSs3526idZ8AuTo7Tu6KPAqv7D4",
      "y": "4EtL6SRW2YlUR5vfVHup7x8PxttmWWhbM4IfyM",
      "d": "870MB6gfutfJ4HtUnUVYMYJpr5eUZNP4Bk43bVdj3eAE",
      "use": "enc",
      "kid": "1"},
    {
      "kty": "RSA",
      "n": "0vx7agoebGcQSuuPilJXZptN9nndRQmbXEPs2aiAFbWhM78LHwx4cbbfAATv8zru1RK7aPFFxuhfDR16tSoc_BJECPeBwKRkJbZCiFV4n3onjhmMstn64tZ_2w-5JsGY4Hy5n9y8XARwl931qt7_RN5w6Cf0h4QyQ5v-65yGjQR0_FDW2Qvq3Y368QQMcTAqqs0KJ2ygnYb9c70zdAZHzu6gMqvRL5hajrnn91cb0pIBD80qNYrLrkt-bfTWnlIAvMQH6e2u0fM41Fd2NCrrwr3XPksINhAQ-G-xBniqbw0Ls1jF44-cSFcur-KeGU8awaJzKnqDKgw",
      "e": "AQAB",
      "x": "X4cTcTeJY Gn4FPYPSbB8rdXix5vwsg1FLN5E3EaG6RJoVH-HLAKD9M7dx5oo7GURknhnRweUkC7hT5f3LM0wBFu0A6fMlwx7V6NqXZuxV70_Ysfqjwp3R7TiBaCxpwp4d0Fk5h2086y_nHNKroAIDk4Ge6pRuohsXywbcReAyMwFs9tv8d_cPVY3i07a3t8MN6TNwmd0Sawm9v47Uic13Sk5ZlG7xojPL4u4b1gU2jx4IBTBNzbWJSzF0H66J78bgkuqsk0GjskDj192Z4qjwbsnn4j2WBi3RL-U521GvY8fKZfmei1z0HIKfzfQ6qm0QYtq0X4jfcKoACQ",
      "p": "83i-71vMGXtoMOJXckv73TKr8637Fi07Z7zy8oj6pbWUQyLPoBQxtPVnwD2OR-60eTDmD2ujmT5pPqMrm8RfmNhVWTdjjMmCMJOpSxicFhj7X0uVIYqyQvW1e6nD363GZY9k3NzR8Ch9vY41xxy89B92Z0GGVQxZxVNe7m350nVbfs",
      "q": "3df0R9cuYq-0s-mkFzLzgltGMeffB2z3hWeMuG0Ocunb33vobPyoumjqvJZQ0idRdwGtCdpYzc0Fw5370ADFxi6Wf_NGEiovoninzKp09VVS7TzFgXkIdrecRzesZ-1kYd_s1qDbxtkDesfAItAG9LUnAdun4Vicb6ye1xk",
      "dp": "G4sPxc6y9y80JW9_Lj4xuppuoluI_zH7VTkS8xj5Sd3XcoEOiMxwYi26mTaue00Ua5dpgF6yBj4c8tQ2Vf4O2RugKDT8pYhF05tAA77Qe_NmtuYzc3C3m3242GqvR5sSDxUXYAN2zq8LFn9EUms6R30b8Ye1K1b7o",
      "d": "s91AH9f9gBsoFR80ac2R_E2gw282rT2k8AOHV1i1mTET1efrA6hUUvMFBcmnp1qew6zvznYY5SSQf7pMDc_ag13nG8Ibp1Bu8JUiraRNqULcQhB_49GF4dHe74WbRsobRonuYTJN1cXpA6T061jvWrX-L186xXw9494Q_cgk",
      "qi": "qYm_p63JRxy5iz1toFgKbWV-JdI3jQ4epy9rbMWx3rQJBfmt90fOYzgUIZEVFECqoweRn81zoDAaa-Bk0KWNdGdJHrZDdmFHw3AN7tI-puxK_mH7ZGJ11ryr8055XlSe3SPmRfKwZi6yU24ZxvQKfYItldUKGz06Ia6zTKVAVRU",
      "alg": "RS256",
      "kid": "2011-04-29"
    }
]}
```
A.3. Example Symmetric Keys

The following example JWK Set contains two symmetric keys represented as JWKs: one designated as being for use with the AES Key Wrap algorithm and a second one that is an HMAC key. (Line breaks are for display purposes only.)

```json
{"keys":
  [
    {
      "kty":"oct",
      "alg":"A128KW",
      "k":"GawgguFyGrWkav7AX4VKug"},
    {
      "kty":"oct",
      "alg":"HMAC",
      "k":"AyM1SysPpbyDfgZld3umj1qzKObwVMk0qQ-EstJQLr_T-1qS0gZHV75aKlMN3Yj0iPS4hcgUUtjAz1z9CAow",
      "kid":"HMAC key used in JWS A.1 example"
    }
  ]
}
```

Appendix B. Example Use of "x5c" (X.509 Certificate Chain) Parameter
The following is an example of a JWK with a RSA signing key represented both as an RSA public key and as an X.509 certificate using the "x5c" parameter:

```json
{"kty":"RSA",
"use":"sig",
"kid":"1b94c",
"n":"vrj0fz9Ccdgx5nQudyhdoR17V-IubWMw7j0hgAszJ_pqYw08PLbK_PdiVGKPrqzmDISfI7sA2S5VENHuiCLNwBuUic011_-7dYbsr4iJmGQ
u2j8DsVY1azpJC_NG84Ty5KKthuCacPod7i7w0L3KorSMHBEwzZDCxTg4a
YWAc8cht-emd9g9q0Vw5MVV42BKSsRngh6X5bUvLYy6AyHKv-J-nUy1wqjyQQDw
H37pl10LtU-0-8SNz1tmRoGE9uJkBLdz5h5FENabwU5mIzQzPdwS-qo-meMV
VfJb6jJVWQpl2SUtCnYG232qvbWbjz_JBP5euqnsIo1vQ",
"e":"AQAB",
"x5c":
["MIIDQjCCAIqgAwIBAgIGATz/FuLiMA0GCSqGSIb3DQEBBQUAMIGIx
CzAJBgNVBAgTAlVTMQswCQYDVQQIEwJDTzEPMA0GA1UEBxMGRGVudmV
aMRAwHgYDVQQKEw5CcmlhbiBDYW1wYmVsbDCCASIwDQYJKoZIhvcN
aBAAQEAJgIBAgIBATz/FuLiMA0GCSqGSIb3DQEBBQUAMIGIxCzAJ
BgNVBAgTAlVTMQswCQYDVQQIEwJDTzEPMA0GA1UEBxMGRGVudmV
aMRAwHgYDVQQKEw5CcmlhbiBDYW1wYmVsbDCCASIwDQYJKoZIhvcN
aBAAQEAJgIBAgIBATz/FuLiMA0GCSqGSIb3DQEBBQUAMIGIxCzAJ
BgNVBAgTAlVTMQswCQYDVQQIEwJDTzEPMA0GA1UEBxMGRGVudmV
aMRAwHgYDVQQKEw5CcmlhbiBDYW1wYmVsbDCCASIwDQYJKoZIhvcN
aBAAQEAJgIBAgIBATz/FuLiMA0GCSqGSIb3DQEBBQUAMIGIxCzAJ

Appendix C.  Example Encrypted RSA Private Key

This example encrypts an RSA private key to the recipient using "PBES2-HS256+A128KW" for key encryption and "A128CBC+HS256" for content encryption.

NOTE: Unless otherwise indicated, all line breaks are included solely for readability.
C.1. Plaintext RSA Private Key

The following RSA key is the plaintext for the authenticated encryption operation, formatted as a JWK object:

```
{
  "kty": "RSA",
  "kid": "juliet@capulet.lit",
  "use": "enc",
  "n": "t6Q8yPswS11djxg3978Th3yNyF1vadM7Df1w9mWep0JhJ66w7amy0K1gPnqFMSQ7y01256Po-TEKodhrriuijxHV78F3V0157aw5WCggPrcaAdzGcS0-Rq-qom-QfcNP8Sjg086MwoqQU_LYw1A9321wSDS_PERYGFlnn3jQ1Q4yBi5sjuCtLrCl4WHL0Pb1fE4v5AuRlUfVcyPbSWyDy8vYjGDs-AqWS92Ia2lGt-GqUmipg0XOC8c20rgxe2ymLHjpHciCKVAbY5-l32-lSeZ0-086U15_aXrKr96gw8cPuAUX1_l8sLGuSiVdt3C_Fn2PZ3Z8744FPFGGcG1sqs2Wz-Q",
  "e": "AQAB",
  "d": "GRtbiQwh9TzysfkgDg4u_N-R_mZGU_9k7JQ_jn1DnfTuMsINprTeaSTyWfS
  NkuaAwn0EbiQy1IQbWv52NY3ybc_iHwJtfr17bAXYEREwaC3hd1PKxy9U
  vqPYGR0kIXTQrRns-dvJ7jahi1I7lykrgpTmrM8dwBo4_PManennyPiQq00nxu
  Toxurt4Zj3v4Qo4xka3GQR09dScZ2ZvsUDmsXOfUENOyMqAC6p1M3h33tsu
  rY15k9qMSpG9OX_IJAXmxzAht_wiZ0w2k4yxHt9S3Lq1yX8C1EwmeRgdk2a
  hec685-oklKt5EPwHkmj01_gj6d9qgc9N6X52esAq",
  "p": "2rnSO4hKSN8sS4qcQFHbs0B0fQKm3sc3c3gh3GRxFrTmdl12ZK9uw-PIHF
  QP6FkXxVrxr-WE-ZEbrqivh_2l1CLS7wA16xVaRt1KkIaVXPPsyBy9yk31s0Q8
  UK96E3_0rADAYTAJs-M3JxClFngqh56HDnETQh3rCT5T3yJws",
  "q": "lu_RiFDp7LbyH3N4GjLr90pSKYP0uQYia2wB0tOCBNjGqxa10RwjsZu0c6I
  edis4S7b_coSKB0kj9pAaBzg-Iy5RvvcQuPamu6rUmjhVt6GT1v6LCKY
  rY15zizqKE-e_ymQknwusUx7eYTB7bAHRK9GqocDESbo808I4s",
  "dp": "KkMTwqBuEwfVw2Z_Dbj1pPqqyHS7j90L5x_MOzqYAJMcLMZtbUtWkvQVDQ3
  tbEo3ZCohbDtt65BfmWzgq6apxQXuBpo0Ou_ahGhxMxKl_hqgi4Q_y_ks1w
  Y521iwrUin5gr7-yO1h41KR-vz2pYHeEaYrhttWtxVqLCRVID6c",
  "dq": "AvfS6-gRQvnm0Bw0JoMsnFseyCkL1wEuEjQFlulUmgwGabQBWtfzI71tkDk
  8GTB9yqPDoYaNO6H7CfrxhJIBQaj6nkF5KKS3TQtQ5qCzhokmxe3KBby
  mXkb5wqUpX5ELD5xcF6FeiaTfWY63TmmEau_lRFCOJ3sDea-ots",
  "qi": "l5Q7i-wNCpyuRueMFRp1sRBk7twN0vSeQpPqmuMqvw57NBuczScEoPswmUqq
  abu9V6-Pr4q57_bapoKRU9R0bvuFnu63SHWEGf1QYJMEavmjdsm-Fp0o
  Yu_neotqQ0zhbI5gry7ajdYy9-2lnX7a9Bzo0Uue9HCJ-UosfS0I8"
}
```

The octets representing the Plaintext used in this example (using JSON array notation) are:

```
```
The following example JWE Protected Header declares that:

- the Content Encryption Key is encrypted to the recipient using the PSE2-HS256+A128KW algorithm to produce the JWE Encrypted Key,
- the Salt Input ("p2s") value is [217, 96, 147, 112, 150, 117, 70, 247, 127, 8, 155, 137, 174, 42, 80, 215],
- the Iteration Count ("p2c") value is 4096,
- authenticated encryption is performed on the Plaintext using the AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext and the Authentication Tag, and
- the content type is application/jwk+json.

```
{  
  "alg":"PBES2-HS256+A128KW",  
  "p2s":"2WCTcJZ1Rvd_CJuJripQ1w",  
  "p2c":4096,  
  "enc":"A128CBC-HS256",  
  "cty":"jwk+json"  
}
```

Encoding this JWE Protected Header as BASE64URL(UTF8(JWE Protected Header)) gives this value (with line breaks for display purposes only):

```
eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJwMnMiOiIyV0NUY0paMVJ2ZF9DSnVKcmlwUTF3IiwicDJjIjo0MDk2LCJlbmMiOi1jBMTI4Q0JDLUhTMjU2Iiw1Y3R5IjoiandrK2pzb24ifQ
```

**C.3. Content Encryption Key (CEK)**

Generate a 256 bit random Content Encryption Key (CEK). In this example, the value (using JSON array notation) is:

```
```
C.4. Key Derivation

Derive a key from a shared passphrase using the PBKDF2 algorithm with HMAC SHA-256 and the specified Salt and Iteration Count values and a 128 bit requested output key size to produce the PBKDF2 Derived Key. This example uses the following passphrase:

Thus from my lips, by yours, my sin is purged.

The octets representing the passphrase are:


The Salt value (UTF8(Alg) || 0x00 || Salt Input) is:

[80, 66, 69, 83, 50, 45, 72, 83, 50, 53, 54, 43, 65, 49, 50, 56, 75, 87, 0, 217, 96, 147, 112, 150, 117, 70, 247, 127, 8, 155, 137, 174, 42, 80, 215].

The resulting PBKDF2 Derived Key value is:

[110, 171, 169, 92, 129, 92, 109, 117, 233, 242, 116, 233, 170, 14, 24, 75]

C.5. Key Encryption

Encrypt the CEK with the "A128KW" algorithm using the PBKDF2 Derived Key. The resulting JWE Encrypted Key value is:

[78, 186, 151, 59, 11, 141, 81, 240, 213, 245, 83, 211, 53, 188, 134, 188, 66, 125, 36, 200, 222, 124, 5, 103, 249, 52, 117, 184, 140, 81, 246, 158, 161, 177, 20, 33, 245, 57, 59, 4]

Encoding this JWE Encrypted Key as BASE64URL(JWE Encrypted Key) gives this value:

TrqX0wuNUfDV9VPTNbyGvEj9JMjefAVn-TR1uIxr9p6hsRQh9Tk7BA

C.6. Initialization Vector

Generate a random 128 bit JWE Initialization Vector. In this example, the value is:

[97, 239, 99, 214, 171, 54, 216, 57, 145, 72, 7, 93, 34, 31, 149, 156]
Encoding this JWE Initialization Vector as BASE64URL(JWE Initialization Vector) gives this value:

Ye9j1qs22DmRSAdIh-VnA

C.7. Additional Authenticated Data

Let the Additional Authenticated Data encryption parameter be ASCII(BASE64URL(UTF8(JWE Protected Header))). This value is:


C.8. Content Encryption

Perform authenticated encryption on the Plaintext with the AES_128_CBC_HMAC_SHA_256 algorithm using the CEK as the encryption key, the JWE Initialization Vector, and the Additional Authenticated Data value above. The resulting Ciphertext is:

The resulting Authentication Tag value is:

[208, 113, 102, 132, 236, 236, 67, 223, 39, 53, 98, 99, 32, 121, 17, 236]

Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this value (with line breaks for display purposes only):
Encoding this JWE Authentication Tag as BASE64URL(JWE Authentication Tag) gives this value:

0HFmhOzsQ98nNWJb0vq5iA0

C.9. Complete representation

Assemble the final representation: The JWE Compact Serialization of this result, as defined in Section 7.1 of [JWE], is the string

BASE64URL(UTF8(JWE Protected Header)) || '.' || BASE64URL(JWE Encrypted Key) || '.' || BASE64URL(JWE Initialization Vector) || '.' || BASE64URL(JWE Ciphertext) || '.' || BASE64URL(JWE Authentication Tag).
The final result in this example is:

ejhbGciOiJQQkVTMi1IUitBMITI4Si1cICJwMnMiOiItY0NUTy0paMVJ2Zf9Dsn
VKcmIwUF3IicDjjio0Mdk2LCJlbnmMoiIBMITI4Z8JDLuHtMjU2IwiY3R5Iioid
andrK2pzb24ifQ.
TrqX0uwunufDv9PTnbyGvEj9JmefAvn-TR1uIXr9p6hsRq9h9tk7BA.
Ye9jqiqs22DrmsadRi-Vna.

AwhB8lxrlKjfn02LGEwq27H4tg9fy2xAbFv3p5ZicHpj64QyHc4qqlz3jEmmZtgQo
wIQzJ13jbyHB8lgeP1quJ1hiF6M2HPlGzw8L-mEeQoJvDUTrER07N0erBk8bwvBqYz6g
0k3QDEOg1IgxyV8-FJvNBwqbn1Bckd6_170tSjxV-8D1rpf-3cJcReIe05Ky30134Z-
GoiAc1kC7zB11c_1Ae11P1-wvrtU1GuJyQyQfXakwdd1_098Kap-UgmyWpFruErJ1JP
nbD4ve950wFEMGLOPfl02MnjaTCDwqQokOj_xplQ2vNPZ8i9ulChBoK1llyQFJL2mOWB
w9bH090j-0b00a0as5mmLsvMQMTf1IrEbbTMzHMBZ8EFw9FwWwF0dWQjGKMNhmBZQ-3
LqTc-M6-gWAD08P0DNFp201b2HGizw611EtA8qRyupFlulaJCLIe1DKGeowKhKkK
ho4DKMN5Nbfq2atm9U0P0ldx5peCUTR1gqMv17up5ZXHTJgPDr5b2N731uooCGAU
qHdgGh9OJUV_0bCtOjsh4CF1SjSuDhrVxyX3Hj2X7cWjRzu3_3Y1GyXU6-s3GFPbi
rfqgEipJDBCThPcoCmymrvyYJYHFn1LqBZRotR89598F95BRxqasADY7UgqGwBqBy665
d0zpyTasvxfxCF_c0Mo3A4W_neFakOW_xp6g4EUDjG1jGSXV9cLstlw_00vdaPdIFLHYHe
PyagyHjouQUUgIqTBsUYrwaF06tgBr8h8v8omLFEjMPPjAzuMuhWw6tBdwGkzd-Ts_u
ub9hrp56Q0Wnt5rGUY0nN2c1-TQ1LxXm5ot014XmnoAqBypwIEgSH3Y4zhwbKHB
Pjso6cduwNdYgbPpp-YVf-2NzZ0Dlq10bWQRBHsbPwyZe_pGkgd564LRTqRwC07CC
_CuYUIiElEssPVsMjRX_U4LEc082TiDDdqKoJrRfuKk5qL818NE9osRDs0OsOQFzi
GrBqxdNsNIaAYAmxkos-13xN4qtBvYx85sCE5U_0MqG7CoxZwoMefRDeapU-C0y
ruOvU1ng8l9jKBBKETY2BpgPEgKBcyCsaAIuAcKamSSC9AxEAX0U0HyhtQtlvMks07AEn
HC2-YywX1FkHmoS4Le6EPfsIlmjA6P1NSg9e95GStETXyGAn6b1xZbHtmwrPSc
ro9LWlhWmAAaL_bxyOObFnxUgXk4vzqZBjZ36UTkJ0tTB-JvkWgFwCFsaw5SwCJ600
4jp07d2y7WmFafj2hTeab2yrum07MUbduz-QQN03y0Sy7TSC1vVMem0NJrwf_cJRe
hKTMd1XGvlDpxZCplr7ZQqRFnF5P8P-14MveqVnaCWn90HlmenczGOS-A-wtRnwmiJ1
B1V_vgJRf4Fdpv-4uHkU-4LWfRxtZKcagg3TtDudRo5_qeBqBUUT_VSCqsFC
OmyWkoj531bxthN19hQXGWbL6frrGM6Ww3h2v3k01zn8FwWv17uFwENyRysAFsnWla125
Tpbj9GvAd12H9HnwhzPS5QhpZKNQ3NMDj13Fn8fz00J893Etbm_tnf0Fcb13x3Bj15
Cz-wz1MghvIPgnGMbMdp_Ad9xpSlyAm9Dq1yeVXk-AigWbUL5nuyWsgyxCpx0cJxW7XM
3820UIeBu-MytL-eneqMD7LyxtsVzcbj0TSVRmhYEMIzUAnisg7usQmQAGRGdRie1TJE
SGmjb_4bZq9s6Ve1LkksI0_QDSrABAle5SUY0fz4FZSFVP5MYmPtoctWdcntPtxLgNA
D1BFX-Z9kAmdZQwGfAMsffle0zAoMe419pMESH0J6B4sJGdcKtQXjicxNYDyoZ7F8
H00Bve7rZd6VIw0MxwFkCtatsv_R-GBchB218rgVPsfYwhWVtuBR4HarpzSBufC
r8_c8fc9Z728sQ08jFjOja6L2x0n_ImzFXN6xw0-Ska-QeuvY3Zx_L310Z4Llp-
7Q5fDhOnX0xFv1Xws-D5mDhD3Z0xOp2zr2TppdKTzb9eWvxxUVv1M8019atBFKPKMAOA
v9oma-A-6v5IXuU0-hLW1MhV3nsq-Jav0c4t6URVUuzjN0OnD_CBGGvnH1JtCh1
88LQxsqLHH1u4FZ-U2SGn1xG7j0-ihiTe2ELGrV4v08E1 BosTmf0cx3qgG0p0eOLBD
IHsrdZ_CCAiTC0HVkMbyq1M6qEhm-q5P6y1QICirwg.
0HFmh0ZsQ98nNwJiHkr7A

Appendix D. Acknowledgements

A JSON representation for RSA public keys was previously introduced by John Panzer, Ben Laurie, and Dirk Balfanz in Magic Signatures
Thanks to Matt Miller for creating the encrypted key example and to Edmund Jay and Brian Campbell for validating the example.

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:


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.

Appendix E. Document History

[[ to be removed by the RFC Editor before publication as an RFC ]]

-36

o Stated that if both "use" and "key_ops" are used, the information they convey MUST be consistent.

o Clarified where white space and line breaks may occur in JSON objects by referencing Section 2 of RFC 7159.

o Specified that registration reviews occur on the jose-reg-review@ietf.org mailing list.

-35

o Used real values for examples in the IANA Registration Templates.

-34

o Addressed IESG review comments by Pete Resnick, Stephen Farrell, and Richard Barnes.

o Referenced RFC 4945 for PEM certificate delimiter syntax.
-33
  o Addressed secdir review comments by Stephen Kent for which
    resolutions had mistakenly been omitted in the previous draft.
  o Acknowledged additional contributors.

-32
  o Addressed Gen-ART review comments by Russ Housley.
  o Addressed secdir review comments by Stephen Kent.

-31
  o No changes were made, other than to the version number and date.

-30
  o Added references and cleaned up the reference syntax in a few
    places.
  o Applied minor wording changes to the Security Considerations
    section.

-29
  o Replaced the terms JWS Header, JWE Header, and JWT Header with a
    single JOSE Header term defined in the JWS specification. This
    also enabled a single Header Parameter definition to be used and
    reduced other areas of duplication between specifications.

-28
  o Revised the introduction to the Security Considerations section.
  o Refined the text about when applications using encrypted JWKs and
    JWK Sets would not need to use the "cty" header parameter.

-27
  o Added an example JWK early in the draft.
  o Described additional security considerations.
  o Added the "x5t#S256" (X.509 Certificate SHA-256 Thumbprint) JWK
    member.
- Addressed a few editorial issues.
- Referenced Section 6 of RFC 6125 for TLS server certificate identity validation.
- Deleted misleading non-normative phrase from the "use" description.
- Noted that octet sequences are depicted using JSON array notation.
- Updated references, including to W3C specifications.
- Updated WebCrypto reference to refer to W3C Last Call draft.
- Corrected the authentication tag value in the encrypted key example.
- Updated the JSON reference to RFC 7159.
- No changes were made, other than to the version number and date.
- Corrected RFC 2119 terminology usage.
- Replaced references to draft-ietf-json-rfc4627bis with RFC 7158.
- Replaced the "key_ops" values "wrap" and "unwrap" with "wrapKey" and "unwrapKey" to match the "KeyUsage" values defined in the current Web Cryptography API [WebCrypto] editor's draft.
- Compute the PBES2 salt parameter as (UTF8(Alg) || 0x00 || Salt Input), where the "p2s" Header Parameter encodes the Salt Input value and Alg is the "alg" Header Parameter value.
- Changed some references from being normative to informative, addressing issue #90.
-20

- Renamed "use_details" to "key_ops" (key operations).

- Clarified that "use" is meant for public key use cases, "key_ops" is meant for use cases in which public, private, or symmetric keys may be present, and that "use" and "key_ops" should not be used together.

- Replaced references to RFC 4627 with draft-ietf-json-rfc4627bis, addressing issue #90.

-19

- Added optional "use_details" (key use details) JWK member.

- Reordered the key selection parameters.

-18

- Changes to address editorial and minor issues #68, #69, #73, #74, #76, #77, #78, #79, #82, #85, #89, and #135.

- Added and used Description registry fields.

-17

- Refined the "typ" and "cty" definitions to always be MIME Media Types, with the omission of "application/" prefixes recommended for brevity, addressing issue #50.

- Added an example encrypting an RSA private key with "PBES2-HS256+A128KW" and "A128CBC-HS256". Thanks to Matt Miller for producing this!

- Processing rules occurring in both JWS and JWK are now referenced in JWS by JWK, rather than duplicated, addressing issue #57.

- Terms used in multiple documents are now defined in one place and incorporated by reference. Some lightly used or obvious terms were also removed. This addresses issue #58.

-16

- Changes to address editorial and minor issues #41, #42, #43, #47, #51, #67, #71, #76, #80, #83, #84, #85, #86, #87, and #88.

-15
o Changes to address editorial issues #48, #64, #65, #66, and #91.

-14

o Relaxed language introducing key parameters since some parameters are applicable to multiple, but not all, key types.

-13

o Applied spelling and grammar corrections.

-12

o Stated that recipients MUST either reject JWKs and JWK Sets with duplicate member names or use a JSON parser that returns only the lexically last duplicate member name.

-11

o Stated that when "kid" values are used within a JWK Set, different keys within the JWK Set SHOULD use distinct "kid" values.

o Added optional "x5u" (X.509 URL), "x5t" (X.509 Certificate Thumbprint), and "x5c" (X.509 Certificate Chain) JWK parameters.

o Added section on Encrypted JWK and Encrypted JWK Set Formats.

o Added a Parameter Information Class value to the JSON Web Key Parameters registry, which registers whether the parameter conveys public or private information.

o Registered "application/jwk+json" and "application/jwk-set+json" MIME types and "JWK" and "JWK-SET" typ header parameter values, addressing issue #21.

-10

o No changes were made, other than to the version number and date.

-09

o Expanded the scope of the JWK specification to include private and symmetric key representations, as specified by draft-jones-jose-json-private-and-symmetric-key-00.

o Defined that members that are not understood must be ignored.

-08
- Changed the name of the JWK key type parameter from "alg" to "kty" to enable use of "alg" to indicate the particular algorithm that the key is intended to be used with.

- Clarified statements of the form "This member is OPTIONAL" to "Use of this member is OPTIONAL".

- Referenced String Comparison Rules in JWS.

- Added seriesInfo information to Internet Draft references.

-07

- Changed the name of the JWK RSA modulus parameter from "mod" to "n" and the name of the JWK RSA exponent parameter from "xpo" to "e", so that the identifiers are the same as those used in RFC 3447.

-06

- Changed the name of the JWK RSA exponent parameter from "exp" to "xpo" so as to allow the potential use of the name "exp" for a future extension that might define an expiration parameter for keys. (The "exp" name is already used for this purpose in the JWT specification.)

- Clarify that the "alg" (algorithm family) member is REQUIRED.

- Correct an instance of "JWK" that should have been "JWK Set".

- Applied changes made by the RFC Editor to RFC 6749's registry language to this specification.

-05

- Indented artwork elements to better distinguish them from the body text.

-04

- Refer to the registries as the primary sources of defined values and then secondarily reference the sections defining the initial contents of the registries.

- Normatively reference XML DSIG 2.0 for its security considerations.
o Added this language to Registration Templates: "This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted."

o Described additional open issues.

o Applied editorial suggestions.

-03

o Clarified that "kid" values need not be unique within a JWK Set.

o Moved JSON Web Key Parameters registry to the JWK specification.

o Added "Collision Resistant Namespace" to the terminology section.

o Changed registration requirements from RFC Required to Specification Required with Expert Review.

o Added Registration Template sections for defined registries.

o Added Registry Contents sections to populate registry values.

o Numerous editorial improvements.

-02

o Simplified JWK terminology to get replace the "JWK Key Object" and "JWK Container Object" terms with simply "JSON Web Key (JWK)" and "JSON Web Key Set (JWK Set)" and to eliminate potential confusion between single keys and sets of keys. As part of this change, the top-level member name for a set of keys was changed from "jwk" to "keys".

o Clarified that values with duplicate member names MUST be rejected.

o Established JSON Web Key Set Parameters registry.

o Explicitly listed non-goals in the introduction.

o Moved algorithm-specific definitions from JWK to JWA.

o Reformatted to give each member definition its own section heading.

-01
- Corrected the Magic Signatures reference.

-00

- Created the initial IETF draft based upon
draft-jones-json-web-key-03 with no normative changes.

Author's Address

  Michael B. Jones
  Microsoft

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