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 for representing 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 August 18, 2014.

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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
   1.1. Notational Conventions
2. Terminology
3. JSON Web Key (JWK) Format
   3.1. "kty" (Key Type) Parameter
   3.2. "use" (Public Key Use) Parameter
   3.3. "key_ops" (Key Operations) Parameter
   3.4. "alg" (Algorithm) Parameter
   3.5. "kid" (Key ID) Parameter
   3.6. "x5u" (X.509 URL) Parameter
   3.7. "x5c" (X.509 Certificate Chain) Parameter
   3.8. "x5t" (X.509 Certificate SHA-1 Thumbprint) Parameter
4. JSON Web Key Set (JWK Set) Format
1. Introduction

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) [I-D.ietf-json-rfc4627bis] data structure that represents a cryptographic key. This specification also defines a JSON Web Key Set (JWK Set) JSON data structure for representing 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 certificate chains, representing certified keys, and 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", "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.

UTF8(STRING) denotes the octets of the UTF-8 [RFC3629] representation of STRING.

ASCII(STRING) denotes the octets of the ASCII [USASCII] 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" and "Collision-Resistant Name".

These terms are defined for use by this specification:

- **JSON Web Key (JWK)**
  - A JSON object that represents a cryptographic key.

- **JSON Web Key Set (JWK Set)**
  - A JSON object that contains an array of JWKs as the value of its keys member.

3. JSON Web Key (JWK) Format

A JSON Web Key (JWK) is a JSON object. The members of the object represent properties of the key, including its value. 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 specific to the kind of key being represented. 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; recipients 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 7.1 or be a value that contains a Collision-Resistant Name.

3.1. "kty" (Key Type) Parameter

The **kty** (key type) member identifies the cryptographic algorithm family used with the key. 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 7.1.
3.2. "use" (Public Key Use) Parameter

The use (public key use) member identifies the intended use of the public key. The use parameter is intended for use cases in which it is useful to distinguish between public signing keys and public encryption keys. It is not intended for use cases in which private or symmetric keys may also be present.

Values defined by this specification are:

- sig (signature)
- enc (encryption)

Other values MAY be used. Public Key Use values can be registered in the IANA JSON Web Key Use registry defined in Section 7.2. 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 key use designation for the first key is desired, the enc (encryption) key use value SHOULD be used, since key wrapping is a kind of encryption. The enc value SHOULD also be used for public keys used for key agreement operations. (The alg member can be used to specify the particular cryptographic operation to be performed, when desired.)

3.3. "key_ops" (Key Operations) Parameter

The key_ops (key operations) member identifies the operations(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 signature or MAC)
- verify (verify 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. Key operation values can be registered in the IANA JSON Web Key Operations registry defined in Section 7.3. 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.

The use and key_ops JWK members SHOULD NOT be used together. Applications should specify which of these members they use, if either is to be used by the application.

3.4. "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.

### 3.5. "kid" (Key ID) Parameter

The `kid` (key ID) member can be used to match a specific key. This can be 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.

### 3.6. "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 [RFC1421]. 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 3.1 of HTTP Over TLS [RFC2818]. Use of this member is OPTIONAL.

While there is no requirement that members other than those representing the public key be populated when an `x5u` member is present, 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 needs to allow for only a subset of the usages that are permitted by the certificate. Similarly, if the `alg` member is present, it should represent an algorithm that the certificate allows.

### 3.7. "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, members other than those representing the public key may also be populated when an `x5c` member is present. 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 3.6 for additional guidance on this.

### 3.8. "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]. The key in the certificate MUST match the public key represented by other members of the JWK. Use of this member is OPTIONAL.

If, in the future, certificate thumbprints need to be computed using hash functions other than SHA-1, it is suggested that additional related JWK parameters be defined for that purpose. For example, it is suggested that a new **x5t#S256** (X.509 Certificate Thumbprint using SHA-256) JWK parameter could be defined by registering it in the IANA JSON Web Key Parameters registry defined in **Section 7.1**.

As with the **x5u** member, members other than those representing the public key may also be populated when an **x5t** member is present. 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 3.6** for additional guidance on this.

### 4. JSON Web Key Set (JWK Set) Format

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

The member names within a JWK Set MUST be unique; recipients 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 7.4** 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.

#### 4.1. "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. This member MUST be present in a JWK Set.

### 5. String Comparison Rules

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

### 6. Encrypted JWK and Encrypted JWK Set Formats

JWKs containing non-public key material will need to be encrypted in some contexts to prevent the disclosure of private or symmetric key values to unintended parties. 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.
JWK Sets containing non-public key material will similarly need to be encrypted. 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.

See Appendix C for an example encrypted JWK.

7. IANA Considerations

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

Values are registered with a Specification Required [RFC5226] after a two-week review period on the [TBD]@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 [TBD]@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for access token type: example"). [Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: jose-reg-review.]

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@iesg.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 makes sense.

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).

7.1. 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 3. 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.

7.1.1. Registration Template

Parameter Name:

The name requested (e.g., "example"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the
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 \texttt{kty} values that the Parameter Name is used with are disjoint; for the purposes of matching \texttt{kty} values, "*" matches all values.

**Parameter Description:**
Brief description of the parameter (e.g., "Example description").

**Used with \texttt{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 \texttt{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.

**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.

### 7.1.2. Initial Registry Contents

- **Parameter Name:** \texttt{kty}
  - **Parameter Description:** Key Type
  - **Used with \texttt{kty} Value(s):** *
  - **Parameter Information Class:** Public
  - **Change Controller:** IESG
  - **Specification Document(s):** Section 3.1 of [[this document]]

- **Parameter Name:** \texttt{use}
  - **Parameter Description:** Public Key Use
  - **Used with \texttt{kty} Value(s):** *
  - **Parameter Information Class:** Public
  - **Change Controller:** IESG
  - **Specification Document(s):** Section 3.2 of [[this document]]

- **Parameter Name:** \texttt{key_ops}
  - **Parameter Description:** Key Operations
  - **Used with \texttt{kty} Value(s):** *
  - **Parameter Information Class:** Public
  - **Change Controller:** IESG
  - **Specification Document(s):** Section 3.3 of [[this document]]

- **Parameter Name:** \texttt{alg}
  - **Parameter Description:** Algorithm
  - **Used with \texttt{kty} Value(s):** *
  - **Parameter Information Class:** Public
  - **Change Controller:** IESG
  - **Specification Document(s):** Section 3.4 of [[this document]]

- **Parameter Name:** \texttt{kid}
  - **Parameter Description:** Key ID
  - **Used with \texttt{kty} Value(s):** *
  - **Parameter Information Class:** Public
  - **Change Controller:** IESG
  - **Specification Document(s):** Section 3.5 of [[this document]]
7.2. 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 3.2.

7.2.1. Registration Template

Use Member Value:
The name requested (e.g., "example"). 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., "Example description").

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.

7.2.2. Initial Registry Contents

- Use Member Value: sig
  - Use Description: Signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 3.2 of [[ this document ]]

- Use Member Value: enc
  - Use Description: Encryption
  - Change Controller: IESG
  - Specification Document(s): Section 3.2 of [[ this document ]]
7.3. 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 3.3.

7.3.1. Registration Template

Key Operation Value:
The name requested (e.g., "example"). 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., "Example description").

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.

7.3.2. Initial Registry Contents

- Key Operation Value: sign
  - Key Operation Description: Compute signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]

- Key Operation Value: verify
  - Key Operation Description: Verify signature or MAC
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]

- Key Operation Value: encrypt
  - Key Operation Description: Encrypt content
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]

- Key Operation Value: decrypt
  - Key Operation Description: Decrypt content and validate decryption, if applicable
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]

- Key Operation Value: wrapKey
  - Key Operation Description: Encrypt key
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]

- Key Operation Value: unwrapKey
  - Key Operation Description: Decrypt key and validate decryption, if applicable
  - Change Controller: IESG
  - Specification Document(s): Section 3.3 of [[ this document ]]
7.4. 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 4.

7.4.1. Registration Template

Parameter Name:
The name requested (e.g., "example"). 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., "Example description").

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.

7.4.2. Initial Registry Contents

- Parameter Name: keys
- Parameter Description: Array of JWK values
- Change Controller: IESG
- Specification Document(s): Section 4.1 of [[ this document ]]

7.5. Media Type Registration

7.5.1. 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], which can be used to indicate, respectively, that the content is a JWK or a JWK Set.

- Type Name: application
- Subtype Name: jwk+json
8. Security Considerations

All of the security issues faced by any cryptographic application must be faced by a JWS/JWE/JWK agent. Among these issues are protecting the user’s private and symmetric keys, preventing various attacks, and helping the user avoid mistakes such as inadvertently encrypting a message for the wrong recipient. The entire list of security considerations is beyond the scope of this document, but some significant considerations are listed here.

One should place no more trust in the data associated with 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.

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.

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.

The security considerations in XML DSIG 2.0 [W3C.CR-xmldsig-core2-20120124], about key representations also apply to this specification, other than those that are XML specific.

The TLS Requirements in [JWS] also apply to this specification.
9. References

9.1. Normative References


9.2. 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.)

```json
{"keys": [
  {
    "kty": "EC",
    "crv": "P-256",
    "x": "MKBCTNiKUSDii11ySs35261DZ8AiTo7Tu6KPAqv7D4",
    "y": "4Et16SRWY1UlUrN5vfvVUhph7x8PxlTMw1lbM4IFYM",
    "use": "enc",
    "kid": "1",
  },
  {
    "kty": "RSA",
    "n": "0vx7aogoBeGcQSuuPlJXZptN9ndrQmbXEps2aiAFbWhM78Lhxw4cbbfAAVT86zwu1R7kApFFxuhDR16tSoC_BJECpeWKRKjBZCI4F4n3okjhnMs
    964z_T_2W-5J5sGY4Hc5ny9yBXArw1931qt7_RN5w6Cf0h4QyQ5v-65YgjQR0_FD2Q
    Vzqy368Q0QIcAtaqs8KJ2gYuBc7d0zgdAZHu66MQvR35hajrn19ICb09I
    SD08gNLyrdkt-bFTwHAi4vMQFh6Wezu0fM41Fsd2NcRwr3XPKsINHaQ-G_xBniIqb
    w0LSj1F44-csFCur-eKgU8awapJzKnqDkgw",
    "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": "MKBCTNiKUSDii11ySs35261DZ8AiTo7Tu6KPAqv7D4",
    "y": "4Et16SRWY1UlUrN5vfVUhph7x8PxlTMw1lbM4IFYM",
    "d": "870MB6gfuTJ4htUnUVMyMjpr5eUZNp4Bkd3bV3eAE",
    "use": "enc",
    "kid": "1",
  },
  {
    "kty": "RSA",
    "n": "0vx7aogoBeGcQSuuPlJXZptN9ndrQmbXEps2aiAFbWhM78Lhxw4cbbfAAVT86zwu1R7kApFFxuhDR16tSoC_BJECpeWKRKjBZCI4F4n3okjhnMs
    964z_T_2W-5J5sGY4Hc5ny9yBXArw1931qt7_RN5w6Cf0h4QyQ5v-65YgjQR0_FD2Q
    Vzqy368Q0QIcAtaqs8KJ2gYuBc7d0zgdAZHu66MQvR35hajrn19ICb09I
    SD08gNLyrdkt-bFTwHAi4vMQFh6Wezu0fM41Fsd2NcRwr3XPKsINHaQ-G_xBniIqb
    w0LSj1F44-csFCur-eKgU8awapJzKnqDkgw",
    "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": "GawgguFyGrwkWav7AX4VKUg"
    },
    {
      "kty": "oct",
      "k": "AyM1SysPpbyDfgZld3umj1qZKbwVMkoqQ-EstJQLR-T-1qS0gZHe75aKTmN3Yj0iPS4hcugUtUwzjAzrI9CAow",
      "alg": "RS256",
      "kid": "2011-04-29"
    }
  ]
}
```

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": "vrjOfz9Ccdgx5nQudylhdoR17V-IubWMe0ZCwx_jj9h0gAsz2J-pqYW08PLDB_PdiVGKpzmDI6ls7as25VNHiUuCLNbwuiG0111-7ydr4il1jMOQ
uZn9pdOtLzppJc_NG84T55KhucaPod71I7wOL9orSMhBEawZDxcTwq4a
YWAwch8t-emd9oVwVbMDc2BXksRngh6X5buULYl65AyHKvJ-nuyw1igzjYQDwH
MTPlCl0Ltu-0-8Snzn1tmR06E9u3kBlhd9h5FENabwunUSn1zqZPdwS-qo-meMv
Vfj6jbjJWpr12SuTcnYhG2c32qvbWbj_Z_jBD5eunqSo1oV",
  "e": "AQAB",
  "x5c": [
    "MIIDQjCCAiqgAwIBAgIGATz/FuLiMA0GCSqGSIb3DQEBAQUAA4GNCAQAwIG
    vQKExNQaW5nIz1mR06E9u3kBlhd9h5FENabwunU5n1zqZPdwS-qo-meMv
    Vfj6jbjJWprp12SuTcnYhG2c32qvbWbj_Z_jBD5eunqSo1oV",
    "MIIDQjCCAiqgAwIBAgIGATz/FuLiMA0GCSqGSIb3DQEBAQUAA4GNCAQAwIG
    vQKExNQaW5nIz1mR06E9u3kBlhd9h5FENabwunU5n1zqZPdwS-qo-meMv
    Vfj6jbjJWprp12SuTcnYhG2c32qvbWbj_Z_jBD5eunqSo1oV"
  ]
}
```
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 encryption operation, formatted as a JWK object:

```
{
  "kty": "RSA",
  "kid": "juliet@capulet.lit",
  "use": "enc",
  "n": "t6Q8PWSi1dkJJ9hTP8hNvF1wadM7dfL9wMep0jHj66w7nyoK1gPNqFMSQryO125gP-TEkOhdWr0uijjHvX7BC0v1IS4w5ACgGpRcAd6zCR0-iQom-QCFNP8Sjg086MwqXQ_1vYw1A2G2WsdS_PERYGFInN3QJO8bYns5jCaLCrwLHLOB0fIeVz4uuFvdCPwS7yvDyY3yJGDSM-AqWS9iZQZilgT-Qguip0XOC0Cc29rgLe2ymHLhpc1vKvA5Ys-L32-1Se0-0s6tUIa_rxrk96w8cPuA1xv1v8SlouSviDzct3C_Fn2P2Z3281744FPFGG0cG6vs2Wz-Q",
  "e": "AQAB",
  "p": "2rn5QvH0zEzYszgKd4gu_N-R_mZU9_97QJ_jn10fntTuMsdNprTeaSTyWfSNkuuAv0OeB1Qy1IQbWv25NY3ybc_iHuUJtfr17bAXYERewaC13idhJKxy9uVqyPYGR0kIXTQRquNs-dvJ7jahl1l7YkrcprrMrMDwDbwB4_PMaenNnPi90q09nxnuToutrZJfjVg40x4ka3G0Rq9dCsezV2vsUDmsx0FeUEnOyMqAcd6plM3h3ttsuwy15k9qMsp9G0Xb_IAJmzmxAh_tw1w0zk2k4xYh9t53lqyXC1EwmeRdK2ahecg85-okLqT55ePwHmhGnqCn96X5sesA",
  "q": "2rnS0v4khKSN8s4CgcQHbfs08XboFdQKum3sc4h3GxrmrTmQd1iZK9u-PWfQP0fKvXvR-xE-zeBzrquvH_21CiLUS7wA1eXvART1KkIAuXpsPB9Yk1sOQ8UK96E36rDAYA1Ajs-M3JxCLFNgghs56HdEnETQh3rCT5t3yJws",
  "dp": "KkMTWqBUefVwZ2_Dbj1pPQyPiHshj90L5x_M0zqYA1cmCMlzBtUtWkvQvDq3tbEo3ZicbDttSbfmWggaBpxXnuBpo0Ef_a_hGMXK_hlqign4tq_KsW12i5jwnJrGy9Oyhh41KR-vz2pYhEAeyHtTtvXvQlCIRviDc6",
  "dq": "AqVfs-9gvvHvW0wJsmSnFXycKiWmweqJFquMgswGiltQBWt2fr1Er7t1x0kb9n9GQTb9yqoDyaYn06H7CFtrkxHJBqajinNkF5KS3Tqt5qCzkxIme3kRbBmxxBkq5wqux5EL5dxFC6fieafWY63TmMeAru_lRFOC3j3Dea-ots",
  "qi": "LsQ1-w9cyPUIreMerP1RsBkL7k IW0s5E0pPqmuMwqvW7NBzucSeCOpEmUwUqabu9V0-Py4d57_bapokRui1909ovbvuF63SHWEFgLzQvdJAmvjm3sf-Fp0oYu_neotq0hzbI5gr7yajy9-21Nxe76aBzOuU9HC-JUSSOI8"
}
```

The octets representing the Plaintext are:

```
C.2. JWE Header

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,
- the Plaintext is encrypted using the AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext, 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:

\[
eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJwMnMiOiIyV0NUY0paMVJ2ZF9DSnVKcm1wUTF3IiwicDJjIjo0MDk2LCJlbmMiOiJBMTI4Q0JDLUhTMjU2IiwY3R5IjoiandrK2pzb24ifQ
\]

C.3. Content Encryption Key (CEK)

Generate a 256 bit random Content Encryption Key (CEK). In this example, the value 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:

Ye9j1qs22DmRSAAdih-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

Encrypt the Plaintext with AES_128_CBC_HMAC_SHA_256 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:


Encoding this JWE Ciphertext as BASE64URL(JWE Ciphertext) gives this value:

AwhB8lxrlkJFn02LGWeq27H4tg9 fyAzAbFv3p5zicHpj64QqYHc44qqlZ3EmnZTGq0wltqZJl3jyH8B8lGepIqU1jHf6M2Hlpzq8w8l-mee0qyJOvDUTER07Nt0erBk8lwBQyZ6g0k3Q3DEIo1fYxV8-FJvNwbq1N8ck6d_1070ijSHV-8D1rp-3JCRie0S5xy30I34ZG0iAiC1EK2B11c_AE11PII_wvrtiUiiy8y0fQxakWdi_098Kap-UgmyWpfreU3J1Pnbd4Vq95w0efPfiol02mNqotqTDCwIqkoj_1plq2VPnzBjguLcb0kl1yQFJL2m0WBwqhoB90j-080as5m5mlsVsMVTF1IIEBEbTMHBZMBEBEF9WfwF0DWqGKMNhmBzq-3lvqC-M6gwA6G8Pdh0Nfp20i2hGlizgiiEa8GRyUpfluljC1ie1dKG0ewkuKkZnh04DKKM5nbqfg2atmU0P0ldx5peCtUR1gM1V7qupszXTjgpd5b2n731uoocGCAuqhDghgq0j9O_b0cTbdh4SOF1s3dUxhXyX3JHh2x2xihfw2727xYX6-Y-3sGFpi frrfQijEpDBEPcocomnyrwjYHfgn1qBZRotRrRs9q8F95bRXgaqdy7UqGwQ8bw665d0zpTavSvfxf_c0MWAL-neFAdK0W_Pxg64EUDgjG1WsxV9cLstLw_0vovapD1FHLHYHePyaghyHq0ujUqg7BsYwywraF06tg8BHV80lmNFMEDmpjaU2Mh6btBdWgKd-Ts-ub9hxrpj4UsoWt5nr5Gy0uN2N_c1-TQ1xym5oto14MxnoAyBQpwIEghSHY4ZhwKBhBhPjSo0cdwnDygbppb-YYVF-2NZz0D1i010WQBRHsbpWyz_xbgKgD504LrtqRwC70CC_Cyury111EssPvSmMjrx_u4LFEOc82tihpdgjK0jRufKK5rql8nB9E9s0qD0as0ofFQzi GrbrqxsDNSYjAIaxmkkos-13nx4qtubvX85sC5E0_UMqGc70XzWOMPEF0aepuV-c0yrvuINig8i1jIKKExETY28gPkgKBCxAsAUCkAmCC9A18xA0U0YHhtqTl1vMks07AEnhNC2-YzPyx1jFkhMs04LlLe6_ePsfM1mja6P1Insge9CSG65ETETYXAn6bixZbHtmmwPscro90LWvmAaA7_bxyOBFnxugWkt4zzqJ36J2O4T0B-JvwgfWFCFswaSWC7Hje6o4jp07d2NY7WmfA1h2Teabz5wumO8TMhDduZ-Q0N3pYob57TSc1vme0N9JrF_cJRehkTftmd1XgVldpx2Cpl7rZqRQh6JFp-14mEoQvCnaw6n90N0hlemczG0S-A-wrhtwmmjBIv_vg3rF4dpV-4huKQ-4lWFxtzKcggq3 TWctdO_qeBbUUT_VSCGscFoMnyWkoj56b1xthn19qy1GwBglMrfrpR6WHw23vk91iz8FVWi7uFwEnY5afsnWlAIz5TpbJ96vAdl29H9Nhzwbp85NqHnPzkNQ3MJDj13F8nbf20J83b3Etbm_fTqFcbx13b3j15Cz-wW1Mg7Vip6GmMNT_A0P9xSi1yAMd9q1yeYVK-A1gwBUIN5uyWS6ycxp0CJxwX7Xm38zOuIeBu-mytL-eqmdM7lytszcbz0jTSrRhymyUAn5gs7qDQAOGdRqIE17ESEMjy4Zq9s6Ve1LLKs10_QOsrABaLeas655U9yF4Z5SfoV5PfMtcw القDDpcLnlxLGNA D1BF-X2zKdMzQW6FmsFle0zA0Me4I9pMEHSEH4BJS6GdcKtxqjx1CNyDuyoF718LH008vEB7z6dvTlwOMxwkFCkatv_r-GsBh2l8GRPsYvhWluTRharpzSBuFc4r8-c8fco2Z7bsQ081jfj0aja6LzXO_inzFNXU6xwof-Ska-QuyvYz3X_L31ZOX4Llp 7qSgfDoh0nx0Fx1wks-D5mHD3x0Uop2b27ppdKTB9eaw2vxUXV1M80IatBPMKGAov90mA-6vv5xUH0L-1WnHCLq_g0vnswp-Jav0c4t6URVUuzjJNOoNCdBOGvHnjhJTC1188LqslmLH1U04F-2USGlnxG7J0-ihi42eLGrv4v08Es106sTmf0Cxx3qG0gPqe0ELBD IHSrdZ_CCACa1Tc0HvMBqy1M6qEHm-q5P6y1QCIrwg

C.9. Complete representation

Assemble the final representation:

The Compact Serialization of this result is the string BASE64URL(UF8(JWE Protected Header)) || `'.' || BASE64URL(JWE Encrypted Key) || `'.' || BASE64URL(JWE Initialization Vector) || `'.' || BASE64URL(JWE Authentication Tag).

The final result in this example is:
Appendix D.  Acknowledgements

A JSON representation for RSA public keys was previously introduced by John Panzer, Ben Laurie, and Dirk Balfanz in [MagicSignatures]. 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, John Bradley, Brian Campbell, Breno de Medeiros, Joe Hildebrand, Edmund Jay, Ben Laurie, James Manger, Matt Miller, Tony Nadalin, Axel Nennker, John Panzer, Eric Rescorla, Nat Sakimura, Jim Schaad, Paul Tarjan, Hannes Tschofenig, and Sean Turner.

Jim Schaad and Karen O'Donoghue chaired the JOSE working group and Sean Turner and Stephen Farrell 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 ]]
-21

-20

-19

-18

-17

-16

-15

-14

-13

-12

-11
- Stated that when kid values are used within a JWK Set, different keys within the JWK Set SHOULD use distinct kid values.
- Added optional x5u (X.509 URL), x5t (X.509 Certificate Thumbprint), and x5c (X.509 Certificate Chain) JWK parameters.
- Added section on Encrypted JWK and Encrypted JWK Set Formats.
- Added a Parameter Information Class value to the JSON Web Key Parameters registry, which registers whether the parameter conveys public or private information.
- Registered application/jwk+json and application/jwk-set+json MIME types and JWK and JWK-SET typ header parameter values, addressing issue #21.

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

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

- 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.

- 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.

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

- Refer to the registries as the primary sources of defined values and then secondarily reference the sections defining the initial contents of the registries.
- 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."
- Described additional open issues.
- Applied editorial suggestions.
Clarified that kid values need not be unique within a JWK Set.
Moved JSON Web Key Parameters registry to the JWK specification.
Added "Collision Resistant Namespace" to the terminology section.
Changed registration requirements from RFC Required to Specification Required with Expert Review.
Added Registration Template sections for defined registries.
Added Registry Contents sections to populate registry values.
Numerous editorial improvements.

-02

- 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.
- Clarified that values with duplicate member names MUST be rejected.
- Established JSON Web Key Set Parameters registry.
- Explicitly listed non-goals in the introduction.
- Moved algorithm-specific definitions from JWK to JWA.
- 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/