Abstract

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a 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 established by that specification.

Status of This Memo

This is an Internet Standards Track document.

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

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7517.

Copyright Notice

Copyright (c) 2015 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 ................................................. 3
   1.1. Notational Conventions .............................. 3
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. 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 ....................... 12
   8.1.1. Registration Template ............................... 12
   8.1.2. Initial Registry Contents ......................... 13
   8.2. JSON Web Key Use Registry .............................. 15
   8.2.1. Registration Template ............................... 15
   8.2.2. Initial Registry Contents ......................... 15
   8.3. JSON Web Key Operations Registry ....................... 16
   8.3.1. Registration Template ............................... 16
   8.3.2. Initial Registry Contents ......................... 16
   8.4. JSON Web Key Set Parameters Registry ................... 17
   8.4.1. Registration Template ............................... 17
   8.4.2. Initial Registry Contents ......................... 18
   8.5. Media Type Registration .................................. 18
   8.5.1. Registry Contents .................................... 18
9. Security Considerations ......................................... 19
   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
Appendix A. Example JSON Web Key Sets .............................. 25
   A.1. Example Public Keys ...................................... 25
   A.2. Example Private Keys .................................... 25
   A.3. Example Symmetric Keys .................................. 27
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 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 established 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] and JSON Web Encryption [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]. The interpretation should only be applied when the terms appear in all capital letters.

BASE64URL(OCTETS) denotes the base64url encoding of OCTETS, per Section 2 of [JWS].
UTF8(STRING) denotes the octets of the UTF-8 [RFC3629] representation of STRING, where STRING is a sequence of zero or more Unicode [UNICODE] characters.

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

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

2. Terminology

The terms "JSON Web Signature (JWS)", "Base64url Encoding", "Collision-Resistant Name", "Header Parameter", and "JOSE Header" are defined by the JWS specification [JWS].

The terms "JSON Web Encryption (JWE)", "Additional Authenticated Data (AAD)", "JWE Authentication Tag", "JWE Ciphertext", "JWE Compact Serialization", "JWE Encrypted Key", "JWE Initialization Vector", and "JWE Protected Header" are defined by the JWE specification [JWE].

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

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.

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

```json
{"kty":"EC",
 "crv":"P-256",
 "x":"f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tu9uFhVrVEU",
 "y":"x_FEzRu9m36HLN_tue659LnpXW6pCyStikYjKIWI5a0",
 "kid":"Public key used in JWS spec Appendix A.3 example"
}
```

Additional example JWK values can be found in Appendix A.

4. JSON Web Key (JWK) Format

A 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 whitespace 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 established by Section 8.1 or be a value that contains a Collision-Resistant Name.
4.1. "kty" (Key Type) Parameter

The "kty" (key type) parameter 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 established by [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 established by [JWA]; the initial contents of this registry are the values defined in Section 6.1 of [JWA].

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

4.2. "use" (Public Key Use) Parameter

The "use" (public key use) parameter 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 to be used for public keys used for key agreement operations.

Additional "use" (public key use) values can be registered in the IANA "JSON Web Key Use" registry established by 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) parameter identifies the operation(s) for which the key is intended to be used. 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 [W3C.CR-WebCryptoAPI-20141211] 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_ops" (key operations) values can be registered in the IANA "JSON Web Key Operations" registry established by 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) parameter 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 established by [JWA] or be a value that contains a Collision-Resistant Name. The "alg" value is a case-sensitive ASCII string. Use of this member is OPTIONAL.

4.5. "kid" (Key ID) Parameter

The "kid" (key ID) parameter 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) parameter 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 [RFC5280]. 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) parameter 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 (Section 4 of [RFC4648] -- 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) parameter 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) parameter 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. JWK Set Format

A 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 JWKs. This JSON object MAY contain whitespace 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 established by 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, that are missing required members, or for which values are out of the supported ranges.

5.1. "keys" Parameter

The value of the "keys" parameter 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.

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

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register JWK parameter: example").
Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG’s attention (using the iesg@ietf.org mailing list) for resolution.

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

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

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

8.1. JSON Web Key Parameters Registry

This section 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 section 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 Experts state that
there is a compelling reason to allow an exception. 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 Experts state that there is a compelling reason to allow an exception.

Parameter Information Class:
Registers whether the parameter conveys public or private information. Its value must be either Public or Private.

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

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

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 RFC 7517

- 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 RFC 7517
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 RFC 7517

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 RFC 7517

Parameter Name: "kid"
Parameter Description: Key ID
Used with "kty" Value(s): *
Parameter Information Class: Public
Change Controller: IESG
Specification Document(s): Section 4.5 of RFC 7517

Parameter Name: "x5u"
Parameter Description: X.509 URL
Used with "kty" Value(s): *
Parameter Information Class: Public
Change Controller: IESG
Specification Document(s): Section 4.6 of RFC 7517

Parameter Name: "x5c"
Parameter Description: X.509 Certificate Chain
Used with "kty" Value(s): *
Parameter Information Class: Public
Change Controller: IESG
Specification Document(s): Section 4.7 of RFC 7517

Parameter Name: "x5t"
Parameter Description: X.509 Certificate SHA-1 Thumbprint
Used with "kty" Value(s): *
Parameter Information Class: Public
Change Controller: IESG
Specification Document(s): Section 4.8 of RFC 7517

Parameter Name: "x5t#S256"
Parameter Description: X.509 Certificate SHA-256 Thumbprint
Used with "kty" Value(s): *
Parameter Information Class: Public
Change Controller: IESG
Specification Document(s): Section 4.9 of RFC 7517
8.2. JSON Web Key Use Registry

This section 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 section 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 Experts state that there is a compelling reason to allow an exception.

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

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

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

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 RFC 7517

- Use Member Value: "enc"
  - Use Description: Encryption
  - Change Controller: IESG
  - Specification Document(s): Section 4.2 of RFC 7517
8.3. JSON Web Key Operations Registry

This section 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 section 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 Experts state that there is a compelling reason to allow an exception.

Key Operation Description:
Brief description of the key operation (e.g., "Compute digital signature or MAC").

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

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

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 RFC 7517

- Key Operation Value: "verify"
- Key Operation Description: Verify digital signature or MAC
- Change Controller: IESG
- Specification Document(s): Section 4.3 of RFC 7517
o Key Operation Value: "encrypt"
  o Key Operation Description: Encrypt content
  o Change Controller: IESG
  o Specification Document(s): Section 4.3 of RFC 7517

o Key Operation Value: "decrypt"
  o Key Operation Description: Decrypt content and validate decryption, if applicable
  o Change Controller: IESG
  o Specification Document(s): Section 4.3 of RFC 7517

o Key Operation Value: "wrapKey"
  o Key Operation Description: Encrypt key
  o Change Controller: IESG
  o Specification Document(s): Section 4.3 of RFC 7517

o Key Operation Value: "unwrapKey"
  o Key Operation Description: Decrypt key and validate decryption, if applicable
  o Change Controller: IESG
  o Specification Document(s): Section 4.3 of RFC 7517

8.4. JSON Web Key Set Parameters Registry

This section 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 section 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 Experts state that there is a compelling reason to allow an exception.
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 RFC 7517

8.5. Media Type Registration

8.5.1. Registry Contents

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

- 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 RFC 7517.
- Interoperability Considerations: n/a
- Published Specification: RFC 7517
- Applications that use this media type: OpenID Connect, Salesforce, Google, Android, Windows Azure, W3C WebCrypto API, numerous others
- Fragment identifier considerations: n/a
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.

In almost all cases, applications make decisions about whether to trust a key based on attributes bound to the key, such as names, roles, and the key origin, rather than based on the key itself. When an application is deciding whether to trust a key, there are several ways that it can bind attributes to a JWK. Two example mechanisms are PKIX [RFC5280] and JSON Web Token (JWT) [JWT].

For instance, the creator of a JWK can include a PKIX certificate in the JWK’s "x5c" member. If the application validates the certificate and verifies that the JWK corresponds to the subject public key in the certificate, then the JWK can be associated with the attributes in the certificate, such as the subject name, subject alternative names, extended key usages, and its signature chain.

As another example, a JWT can be used to associate attributes with a JWK by referencing the JWK as a claim in the JWT. The JWK can be included directly as a claim value or the JWT can include a TLS-secured URI from which to retrieve the JWK value. Either way, an application that gets a JWK via a JWT claim can associate it with the JWT’s cryptographic properties and use these and possibly additional claims in deciding whether to trust the key.

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, except that the "x5u" JWK member is the only feature defined by this specification using TLS.

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), "Media Types", <http://www.iana.org/assignments/media-types>.

[ITU.X690.1994]


Jones                Standards Track          [Page 22]


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. (Line breaks within values are for display purposes only.)

```
{"keys":
[
  {
    "kty":"EC",
    "crv":"P-256",
    "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
    "y":"4Et16SRW2YiLUrN5vfvVHuhp7x8PxltmWW1bbM4IFyM",
    "use":"enc",
    "kid":"1"},
  {
    "kty":"RSA",
    "n": "0vx7agoebGcQSuuPiLXZptN9nnadrQmbXEps2aiAFbWhM78LhWx4cbbfAAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWRRXjBZC1FV4n3oknjhMs
    tn64tZ_2W-5JsGY4Hc5n9yBXArw193lt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2Qvzy7y368QMicAtaSszs8KJZgnYb9c7d0zgdAHzu6QnQvR5h5ajrn1n91CbOpb1
    SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4IFd2h3r3XPsINHsQ-G_x8niIgb
    w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
    "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 within values are for display purposes only.)
"keys": [
  {
    "kty": "EC",
    "crv": "P-256",
    "x": "MKBCTNiCkUSDi11ySs3526128AiTo7Tu6KPAqv7D4",
    "y": "4Et16SrW2y1UxN5vfVHuhp7x8Pxls4mW1bmm41FYm",
    "d": "870MB6gfutJ47HtUnUVMyJpr5eUZNP4Bk43bVdj3eAE",
    "use": "enc",
    "kid": "1"
  },
  {
    "kty": "RSA",
    "n": "0vx7aogebGcQSuuPiLJXZptN9nndrQmbXEsps2aiAFbWhM78L8wX4cbbfAAATv86zu1RK7aPFFxuhDR1L6tSoc_BJECPe6bWKRXjBC2tFvY4n3oknjhMst64nT2w-5jSyG4Hc5n9yBXArv9l31qt_7RN5w6Cf0h4QyQ5v-65YGjQRO_FD2WQvzqY368QOmQnA5aSg8sK2ZJQynYb9c7d02zgdAZHzu6cQMrVL5han191CIbOpbISD08qNLYrLvt-bFTWhtAI4vMQfhh66xwzu0FM41FD2NCrwr3XPksINhaQ-G-xBniIqbw0LsljF44-cSFUrk-eGug18awapjXkNgDKyw",
    "e": "AQAB",
    "d": "X4cTTejY_gn4FYPsXB8rdXix5wsg1FLN5E3EaG6RJoVH-HLLK09M7d5o07UGRknhrnRweUKc7hT5fLJM0WbFAKNLWY2v7B6NqX5sZuvXT0_YSfqijwp3RZw1BaCwsp4dFoK5n2o8cy_nHKKroADk2J46pRUohsXywBreAdYaMwFs9tv8d_CPVY3j07a3t8MN6Twm0dSawm9v47UI1C3Sk52i7xoJPl4sbgj12jx4jIBTBznvbJSzFHK66b7bkguqsk2G0jxsDkJ1924qjwbsnzn4j2WBii3RL-U21GyK8yFkFzmelz0Hllkf0Y6mgn0Yttc0X4jfcAo8Q",
    "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 within values are for display purposes only.)

"keys":
[
  {"kty":"oct",
   "alg":"A128KW",
   "k":"GawgguFyGrWKav7AX4VKUg"},
  {"kty":"oct",
   "k":"AyM1SysPpbyDfg2ld3umj1qzKObwVMkoqQ-EstJQLr_T-lqS0gZH75aKtMN3Yj0iPS4hcgUuTwjAzr1Z9CAow",
   "kid":"HMAC key used in JWS spec Appendix 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 (with line breaks within values for display purposes only):

```json
{"kty":"RSA",
 "use":"sig",
 "kid":"1b94c",
 "n":"vrj0fz9Cdgdgx5nQudyhdoR17V-IubWMMeOZCwx_X_jj0hgAsz2J_pqYW08Plbk_PdiVGPkprzmdISl7sA25VEnHulucLJWbUiiC011_7-dYbmsrijM6GQ
u2j8DsVT1azpJCG-N8G4t5KThuCqod7iI7w0LKe9orSnhBEwzwZDCxtWq4a
YWAmMv8C8-e+-m9DqOwVtmJvMDC2BxksRngh65bUXJ6yAhKv-j-nUy1w gzjYQdWH
MTp1C0LoU-o-8SnnmRoGE9uJkBLdh5gFENaBwnU5m1z2qPdw5-q-o-meMv
VfJb6jJVWpr12SUtCnYt232qvbWbjZ_JjBP5eunqSio1vQ",
 "e":"AQAB",
 "x5c":
 ["MIIDQjCCAiqgAwIBAgIGIeMwEwYDQYJKoZIhvcNAQEBSAMHAgEAMC\nB8gjZHR0dXBsVzIvU3RvZWdJXzEwMBYwJwYDVQQDEx1Gcm9zc3JvdXB\nQaA1/wgZIhvcNAQEBGgYDVQQIEwZDLwYDVQQKEwZPV29yZSBXaWV\nOT+...XU2SgQ=="]
}
```

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 (with line breaks within values for display purposes only):

```json
{
  "kty": "RSA",
  "kid": "juliet@capulet.lit",
  "use": "enc",
  "n": "t6Q8PSW1ldkJhjTP8hNYF1vadM7DF1w9mWepOjHJ66w7nyoK1gPNqFMSQRy0125GP-TEkodhWr0iujjHV78cVo11S4w5ACGgPrAad6zscR-0jqm-QFcNP85jg086MwoqQU_LYyw1AGZ2lWsdqPeryGFinnj3qQ108YnsqCtLCRwLHL0Pb11Ev45AuRiuufVCPySBYWynDgxyvJGD3M-AqWS9z1QZ11gT-GqUmipxOGC0CC20rge2ymLHjpxHcJicvKVvAYB6-L3Z-1se20-0s6u15-aXrk9Gw8cPuA1I83LGUsiVdt3C_Fn2P3Z28l744FPPFGGcG1qs2Wz-Q",
  "e": "AQAB",
  "d": "GRtbIQmoil0ztyzsfzKd4q4u-N-R-mzGU_9k7JQ_jn1DnfTuMsNnPrTeaSTyWfSNkuawan0EElIQVy11QbWV25NY3ybc_iHUTUfri7bAYXEREwAC13h1dPRKky9UvgYGR0kIXTGQrqs-dVJ7jajh17LyckrpTmrM8DwBo4_PMaenNnPiq900xnuToxutRZfjVg40x4ka3GQrQd9CSc22vsUDmxX0fUENOyMqAC6pC6m33ntsursy15k9qMSp9O1XJAXmxzAh_twI0wzkK4xyhx9tS3lqyX8C1EWmeRDk2ahecG85-oLQk5PVEHMMkoi_gJSDsgqC9N6X52esAQ",
  "p": "2znSOV4hKSN8sS4CcqQHFbs0XbodQKum3sc4h3GrXRmQd1lZK9uw-PFIHFQP0FkkXVrx-WEZEbrq1vH_2iCLUS7wA16XvArTk1k1aUXPPSYB9yj31s0Q8UK96E6_OrADAYTaj-M3JxCLF Ngh56HdnE7TqHh3rCT5T3yJw",
  "q": "1u_R1FDP7LByh3N4GXLr9OpSKYP0quQ2YiaZwbToCBBNJqXaj10RWjszUc6iedis4S7B_coSKB0Kj9PaPaBzg-IySRvvcQuPamQ66rihjVTG61TV8LCYKy152ziqK0E3ym2qknwsUX7eYTB7LbAhR9KqGqD6B0f80I4as",
  "dp": "KkMTWqBEufVwZ2_DbjpTPqySHhj90L5x_MOszYAJcM1MztbUtwKqVD3q3b1eo3C1ohdbDtt65bfnWzgabPqNXuxPBoOoF_a_HgMXX_1hqiqI4y_kqS1wY52Iwunj5rGrJ-yO1h41KR-zv2pyHAEyHrhtWtxVqLCRviD6c",
  "dq": "AvFS0-GRxvOn0bwMoSMXfyKC1XunuEjQF1gMGfWisQWbfz1E7r1tXDbkn9GQT9VypqDyAn06h7cFtrkxJIBQaj6njfK5KKS3TQ7Q5Czk0XmxeI3RrBymXbk5bQu1PE5LD5x66fEiAfWY63TmmeEau_1RFCC0J3xDea-ots",
  "qi": "1Sqi-w9CpyURMePR1RsbBLk7wN0s5E0PqPmQwM57N8cScEoPwMyGq9w90V-py4dQS7-bapokRut190bvuF56S3HWEFGi2QVJDMenAvmjs4m-Fp0oYu_neotGQ0hzb15gry7ajdYy9-21NX_76bzoOu9HCJ-JusfSOI8"
}
```

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

```json
```
C.2. JOSE 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,
- 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):

```
eyJhbGciOiJQQkVTMi1IUzI1NitBMTI4S1ciLCJwMnMiOiIyV0NUY0paMVJ2ZF9DSn
VKcmlwUTF3IiwicDJjIjo0MDk2LCJjbmMiOiJBMtI4Q0JDLYhTMjU2IiwY3R5Ijoi
andrK2pzb24ifQ
```

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:

TrqXOwuNUfDV9VPTNbyGvEJ9JMjefAVn-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:

Ye9j1qs22DmRSAddIh-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):

```
AwhB8lxrlKf02LGWEeqg27H4Tg9fyZAfbFv3p5ZicHpj64qYc44qq1Z3EmnZTgQo
wQjZ1j3yHB8LgePlQjHj6fM2HPigwz8L-MeeQ0JvDUTrE078htCeBk8b8bQyQ26g
0kQ3DE0GIyFyv7iFvNBWBQhBkd6_i70t_jsHv-8D1rP-3jCRIe50YYy30i134Z
G01Ac1EK21Bl1c_AE11Pi1_wvwtiUiUGb80fQXaKwd1_098Cap-UgmyWfFrequJ31JP
nB04vEweFGMLOPfo2MnjaTCwQyOko_j_xpiQ2vNPf38igulcb0k1lyqFJL2mnOB
wqBh90oj-0800as5mmLsvQMFt1rIEBtmZHM28EF9fWwF0DuDQGKmH NBa2Q3-1
vqT-c-Mg6WAD8DP3wOnFp20ib2HGizwGiyEaX8GRyUpUluJCLie1DKoewKhKkZ
04DKNM5Nbufg2atmU09P0Ldx5eCUTGriGmV17qUpz5XH7tgFPr5b2N731UoCCGAT
qHdgGh0JvJ_OcTDxjs4CFL5SdUhrVx3yHJh2Xd7CwJRzu_3Y1GxYU6-s3GFPb
rFqEqIpJDBTHpc0CMyryrJyYHFg0n1BZRotRsrs958F95bRhxsaQy7UGQwQBwQwy
6S0zpytvTsvxf_xCOMWA-leNEFKOW_Px6g4EUDjGWSXV9cLtLw_OvdadPILFYHHe
PyagHjovQuUgiqc7bsywYpwaF06tgb8ghV8omLNFemDMpJaUzMuHw6bCd6BwGKdZ-t
ub9hrxPj4usQ6Mntr5GyGy8n2N_c1-TQ1Xxm50t014Mxn0AxBQbwIeqSH3Y4zhKBH
Pj60ocuwVdYbGFPb-YyVF-2NZoD1Q10wQWBRHShBPWyz_xbGkgD504LRtqRWCo7CC
_CyyURi1EssPvSmJRX_U4LFE0c82tiDdqjoKRufK5rql8nBE9sQQDsaOoaQZi
GFrBqxdNsYNiAYAmmxsk-13X4gtbyVx85sCE5U_0MqG7C0ZXWOPEFRdaePV-DC
rovUIng881JkBxqITYZBqPegKBYCxsAUcAkKamSCC9AIXbAXUOUHhTqT1vMks07AE
hNCA-YzpXyFkhMoSo4LE6e6E_PFsMmJn6PNSeg9c5G7ETYXGAn6blxZfBtmwermPSc
or9LwhVmAmAA7bxY0bnnfU9q7xZL4zq8BjZ36UTk40TB-JvKwfwVCFSaw5WCH6jo6
o4p07d2y7W6Faj2hTeAbz9wumQTMhBdzu-ZON3pYoB5Y7TSCL1vVme0NJRwF_cJRe
hKTFmd1XGV1xZCp1rT7ZqQbCnHJP-14mEVPNc AwGn9ONHlemczGOS-AAwntwnm7J
B1V_ygvJR1f4Fdpv-4hU4k-q-PLpu3-11WXrZTkkgcq3tWtdrouo5_QeobQUBUt_VCSgfC
OmyWkoj563bthhN19hqIOXwIBP6rhMh23vk01zn8Fv8izFWeRISafsaNLz
Tp7bj9GAd12H9HNzpsB5Qh2BZQ3Mmbj13f8e00J883EtmknFqfcb13X3b15
Cz-Wt1MghvIpGmmMBT_Adp9xSiyAM9d1qyeVXK-AlgWBUI5uwyWGSyCpoxCw7wHM
38z0U1eBz-Maylt-eqndMLxysTsvCbj0T3VrhrYEMITAUan5gs7uMQAGRdgRIELTJE
SGmjb_4bZq9s6VeILKs10QDsrABAe55UYy0zF25S0V5PMypLtcwv_dcnPLxgNA
D1BFZX_59AAdMZQ6fAmmsfLe0zAoMe419pMEH0JB4sJGdCkxQxjicKXydyOyoF718
H00BvYr7zdVtIw0MxwkeFCtatxv_S-GbsCCH218RvgPsfYhwuT8R4HarpsbDufC4
r8_c8fc92278v081jFjoA6L2xON_ImZFXU6xwo-Ska-QeuvY23X3L1Z0X4ALp-
7QSfDoHomNsvF1vXws-D5omDMD3xzouOzb2b2TppdkTKZb9eW2vUvIW1M0f91atBFVKMGaO
v90ma-6v55ixUho0-1wM1HLQ_q8vmsnp-Jav0c4t6URVVzu7NOoNd_CBGGVHnijTCHL
81LQxsLHLH1u4Fz-UZSGLnxGT0-ihih2EFLGFr4v08E1BsmTmfox3qggOPq0eOLBD
IHardz_CCAitC0HVkbMybq1MeqEhM-q5P6ylQCIrwg
```

Encoding this JWE Authentication Tag as BASE64URL (JWE Authentication Tag) gives this value:

```
0HFmh0zsQ98nNWJiHKr7A
```
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 (with line breaks for display purposes only) is:

eyJhbGciOiJQQkVTM1I1UzI1NitBMTI4S1ciLCJwMnMiOiV0fNY0paMVJ2ZF9DSn
VKnmlwUTF31licdDJi0jOMDK2LCJ1bmi0i1jBMTI4Q0JDLUH1MjU2I1wiy13R5Ijoi
andrK2pzb24ifQ.
TqXOWnUWfDV9VTNbyGvEJ9JMjeFAVn-TR1uIX9p6hsRq9h7T9BA. 
Ye9j1qsz2DMrSADDhI-VnA.
Awh8BlxrixKJ fn02LGWEqg27H4t9gfy2AbFv3p52icHp6j49yHc44qqlz13EMm2TgQo
w1qZJj3yjHBs8g2ePiUj1hf6M2HPgzw8L-mEeE0q0jUDEUT07nc0eRkk68rwBQy26g
0kQ3DEOi1gfYxY8-FJvNVBYwqbt1Na6c6d170tJSHV-81Drp-3jcRie05KY3oi34Z_
GO1Ac1E1k2B1ic_AE1U1IPi-wvvtRiUg8YofQXakWd1_098Kap-UqmyWFlreUJ31JP
nb4Du95oweEFMLOPlfo2MjnaTDCWQokoJ_xplQ2vNZp8iguUlCHboK1lyQFjL2nOWB
wqhnBo90j-O800ao5mnLSxvQMTflIrIEebbTmZHMZ82EF9fWwwFu0DWWQjGkMNhmBZQ-3
1vqTc-M6-gWA6D8DOnOenP200ib2HgizwGlEAx18GRyUpfluLi1Je1dKQOeohwKukKz
h04DKN5M5nbugf2atmU90P0Ldxs5eCutRGlglMV7op52XHTfgDPr5bN731UooCGAU
qHdgGhG0JVj_OcTdsj5H4F1SJsUhrXvyXy3HJh2c7cwJRzu_3Y13gyXU6-s3GFpB1
rfQqEipJDIBTHpcOcmrywfwYHfgnlqBZRotRrS959f95bRQxsaday7UqGQwBQwByw665
d0zpvTasvxfC_c0nWAlFwAk00U-Px6g4EUDjG11GqG5Yx9LCStLx-OovdApDIFbA
PyagyHjou0QUGiq7BsayYrwF0Htgb88v8omLHFEEnmDPJ2aUZmuHw6tBdGzkD-t_s-
ub9hxpjsJ4UsWnt5rGUyoN2N_c1-jQ1Xxm5o14MxnoAByBqpwIEGr3H42hkbKbH
PjSo0cduwNdYbGpPp-bYvF-0Z2o0DlQ0vWQBRHBsbPwyz_xbKgk5D04LrgtQw0c7CC
_CyyURi1sEssVPsMjRX_U4LPEOc82TidddqK0jRufkk5rql8nBE9soQODSaOsPFQzi
GrBrxdsNYAYAmxmxsk-i3N4qtByVX85sesC5tu_0mgq7COxZM0PEFrdapelpUV-cOy
rv0UIqng818JkBBkEY2ZQgênciaKBCXAuAcKamCSC9AiBxA0U0HyhTqltvMks07AE
hNC2-YzPxy1FkhMoS4LLe6E_PfSm1mja6P1NSge9c5G5tETYXG6n6ixBzHtmwrPFsc
r9OLWhVmVaa7_bx0ObnFuxwWk4vzvQbJz3J6UTk40aJr-JkWqFVCFasaw5Wchj60o
4j0P72yN7WmFajZbEabz9wumQ0THMhBd2uQ-0Q3p0YObSy7TSC1vVme0NjkVf1cRe
hKTFmd1XGv1dpXzPcPr3YQqKQh8F0JF-4mEQVnaCawGn90N1lemczG0S-A-wwtnwMJ
B1V_ygJRf4FdPv-4hUk4-QLpu3-11FwxtZKcgqq3tWTudu05_QebQbUUT_VScgsF
OmyWKjOj56lbxhTn9HIqXGWBIGfrrM6WHD3vkV01zn8F8Vfw7uJwEnRYSaFsnWL1A3Z
TbpI99GVAd12H9NFwbp2NqHPKq3NMBi1j3F8n9f0j883etmbmTmFnQFcb13X3B1j5
Cz=-W1MrHgipvGGnM8T stabbing_s0109And9q1yeyVXk-AILgWBUN5nuyWGSyCxp0C JWx7n
38z0u1eBu-eMyt-eqndm7hLxysaVzCbj0TSVmhyEMYizUsAn1gs7us9MqARGdQREIJTE
SGMjbo_4izqgs9v16lek5s1-o3QdrABase19YUyOZF5SzV5PMyPtcwcv_dcNPILxGnA
D1BFX-Z9kaDmMZQWf6amsflFle0aOaMe419PMEHS0JB4sJGdcKtQxjicXNydYDyzF718
H00Bv0rEr3z6dtvI0mwpkCtsvR-_GsbCH218RgVpsvYHwvUtR4HarpszDsBucF4
r8_cBf9Z22J8sQ801Jf10JoJa6L2x0N_ImzFND0XU60wo-x-BkQevYu23X_3L13Z041Lp-
7Q5fgDoHnx0FxVxws-DsnMDH23xOup2bs2pckTDzh9eW2vUVIvmB19atBFPMKGAO
v9omR-6v5vxIUX0-L1m3HLQ_gvmsnp-Java0c4t6URVuzujNOOnd_CBCGvHvH1Jpcht1
81LQsgqLHHJu4Fz-2UGSN1xGTj0-ih2TELGRV4v08ElBosTmfoxc3xqggQP0eq0LBD
IHsrdZ_CBCAtiOHVkBmyq1ML6qEhM-q5P6y1QCIrwg.
0HFmOzsQ98nNWjIHkR7A
Acknowledgements

A JSON representation for RSA public keys was previously introduced by John Panzer, Ben Laurie, and Dirk Balfanz in Magic Signatures [MagicSignatures].

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.

Author’s Address

Michael B. Jones
Microsoft

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