                    JSON Web Key (JWK) Thumbprint

Abstract

   This specification defines a method for computing a hash value over a
   JSON Web Key (JWK). It defines which fields in a JWK are used in the
   hash computation, the method of creating a canonical form for those
   fields, and how to convert the resulting Unicode string into a byte
   sequence to be hashed. The resulting hash value can be used for
   identifying or selecting the key represented by the JWK that is the
   subject of the thumbprint.

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/rfc7638.

Table of Contents

1. Introduction

   This specification defines a method for computing a hash value
   (a.k.a. digest) over a JSON Web Key (JWK) [JWK]. It defines which
   fields in a JWK are used in the hash computation, the method of
   creating a canonical form for those fields, and how to convert the
   resulting Unicode string into a byte sequence to be hashed. The
   resulting hash value can be used for identifying or selecting the key
   represented by the JWK that is the subject of the thumbprint, for
   instance, by using the base64url-encoded JWK Thumbprint value as a
   "kid" (key ID) value.

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.

2. Terminology

   This specification uses the same terminology as the "JSON Web Key
   (JWK)" [JWK], "JSON Web Signature (JWS)" [JWS], and "JSON Web
   Algorithms (JWA)" [JWA] specifications.

   This term is defined by this specification:

   JWK Thumbprint
      The digest value for a JWK.

3. JSON Web Key (JWK) Thumbprint

   The thumbprint of a JSON Web Key (JWK) is computed as follows:

   1. Construct a JSON object [RFC7159] containing only the required
       members of a JWK representing the key and with no whitespace or
       line breaks before or after any syntactic elements and with the
       required members ordered lexicographically by the Unicode
       [UNICODE] code points of the member names. (This JSON object is
       itself a legal JWK representation of the key.)

   2. Hash the octets of the UTF-8 representation of this JSON object
       with a cryptographic hash function H. For example, SHA-256 [SHS]
       might be used as H. See Section 3.4 for a discussion on the
       choice of hash function.

   The resulting value is the JWK Thumbprint with H of the JWK. The
   details of this computation are further described in subsequent
   sections.

3.1. Example JWK Thumbprint Computation

   This section demonstrates the JWK Thumbprint computation for the JWK
   below (with the long line broken for display purposes only):

```
     {
      "kty": "RSA",
      "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAt
            VT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMstn6
            4tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FD
            W2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n9
            1CbOpbISD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINH
            aQ-G_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
      "e": "AQAB",
      "alg": "RS256",
      "kid": "2011-04-29"
     }
```

   As defined in "JSON Web Key (JWK)" [JWK] and "JSON Web Algorithms
   (JWA)" [JWA], the required members for an RSA public key are:

   o "kty"
   o "n"
   o "e"

   Therefore, these are the members used in the thumbprint computation.

   Their lexicographic order, per Section 3.3, is:

   o "e"
   o "kty"
   o "n"

   Therefore, the JSON object constructed as an intermediate step in the
   computation is as follows (with the line broken for display purposes
   only):

```
     {"e":"AQAB","kty":"RSA","n":"0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2
     aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCi
     FV4n3oknjhMstn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65Y
     GjQR0_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n
     91CbOpbISD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_x
     BniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw"}
```

The octets of the UTF-8 representation of this JSON object are:

[123, 34, 101, 34, 58, 34, 65, 81, 65, 66, 34, 44, 34, 107, 116, 121,
34, 58, 34, 82, 83, 65, 34, 44, 34, 110, 34, 58, 34, 48, 118, 120,
55, 97, 103, 111, 101, 98, 71, 99, 81, 83, 117, 117, 80, 105, 76, 74,
88, 90, 112, 116, 78, 57, 110, 110, 100, 114, 81, 109, 98, 88, 69,
112, 115, 50, 97, 105, 65, 70, 98, 87, 104, 77, 55, 56, 76, 104, 87,
120, 52, 99, 98, 98, 102, 65, 65, 116, 86, 84, 56, 54, 122, 119, 117,
49, 82, 75, 55, 97, 80, 70, 70, 120, 117, 104, 68, 82, 49, 76, 54,
116, 83, 111, 99, 95, 66, 74, 69, 67, 80, 101, 98, 87, 75, 82, 88,
106, 66, 90, 67, 105, 70, 86, 52, 110, 51, 111, 107, 110, 106, 104,
77, 115, 116, 110, 54, 52, 116, 90, 95, 50, 87, 45, 53, 74, 115, 71,
89, 52, 72, 99, 53, 110, 57, 121, 66, 88, 65, 114, 119, 108, 57, 51,
108, 113, 116, 55, 95, 82, 78, 53, 119, 54, 67, 102, 48, 104, 52, 81,
121, 81, 53, 118, 45, 54, 53, 89, 71, 106, 81, 82, 48, 95, 70, 68,
87, 50, 81, 118, 122, 113, 89, 51, 54, 56, 81, 81, 77, 105, 99, 65,
116, 97, 83, 113, 122, 115, 56, 75, 74, 90, 103, 110, 89, 98, 57, 99,
55, 100, 48, 122, 103, 100, 65, 90, 72, 122, 117, 54, 113, 77, 81,
118, 82, 76, 53, 104, 97, 106, 114, 110, 49, 110, 57, 49, 67, 98, 79,
112, 98, 73, 83, 68, 48, 56, 113, 78, 76, 121, 114, 100, 107, 116,
45, 98, 70, 84, 87, 104, 65, 73, 52, 118, 77, 81, 70, 104, 54, 87,
101, 90, 117, 48, 102, 77, 52, 108, 70, 100, 50, 78, 99, 82, 119,
114, 51, 88, 80, 107, 115, 73, 78, 72, 97, 81, 45, 71, 95, 120, 66,
110, 105, 73, 113, 98, 119, 48, 76, 115, 49, 106, 70, 52, 52, 45, 99,
115, 70, 67, 117, 114, 45, 107, 69, 103, 85, 56, 97, 119, 97, 112,
74, 122, 75, 110, 113, 68, 75, 103, 119, 34, 125]

Using SHA-256 [SHS] as the hash function H, the JWK SHA-256
Thumbprint value is the SHA-256 hash of these octets, specifically:

[55, 54, 203, 177, 120, 124, 184, 48, 156, 119, 238, 140, 55, 5, 197,
225, 111, 251, 158, 133, 151, 21, 144, 31, 30, 76, 89, 177, 17, 130,
245, 123]

The base64url encoding [JWS] of this JWK SHA-256 Thumbprint value
(which might, for instance, be used as a "kid" (key ID) value) is:

NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs

3.2. JWK Members Used in the Thumbprint Computation

   Only the required members of a key's representation are used when
   computing its JWK Thumbprint value. As defined in "JSON Web Key
   (JWK)" [JWK] and "JSON Web Algorithms (JWA)" [JWA], the required
   members for an elliptic curve public key for the curves specified in
   Section 6.2.1.1 of RFC 7518 [JWA], in lexicographic order, are:

   o   "crv"
   o   "kty"
   o   "x"
   o   "y"

   The required members for an RSA public key, in lexicographic order,
   are:

   o   "e"
   o   "kty"
   o   "n"

   The required members for a symmetric key, in lexicographic order,
   are:

   o   "k"
   o   "kty"

   As other "kty" (key type) values are defined, the specifications
   defining them should be similarly consulted to determine which
   members, in addition to "kty", are required.

3.2.1. JWK Thumbprint of a Private Key

   The JWK Thumbprint of a JWK representing a private key is computed as
   the JWK Thumbprint of a JWK representing the corresponding public
   key. This has the intentional benefit that the same JWK Thumbprint
   value can be computed both by parties using either the public or
   private key. The JWK Thumbprint can then be used to refer to both
   keys of the key pair. Application context can be used to determine
   if the public or private key is the one being referred to by the JWK
   Thumbprint.

   This specification defines the method of computing JWK Thumbprints of
   JWKs representing private keys for interoperability reasons -- so
   that different implementations computing JWK Thumbprints of private
   keys will produce the same result.

3.2.2. Why Not Include Optional Members?

   Optional members of JWKs are intentionally not included in the JWK
   Thumbprint computation so that their absence or presence in the JWK
   does not alter the resulting value. The JWK Thumbprint value is a
   digest of the members required to represent the key as a JWK -- not
   of additional data that may also accompany the key.

   Optional members are not included so that the JWK Thumbprint refers
   to a key -- not a key with an associated set of key attributes.
   Different application contexts might or might not include different
   subsets of optional attributes about the key in the JWK. If these
   were included in the calculation of the JWK thumbprint, the values
   would be different for those JWKs, even though the keys are the same.
   The benefit of including only the JWK required members is that the
   JWK Thumbprint of any JWK representing the key remains the same,
   regardless of any other attributes that are present.

   Different kinds of thumbprints could be defined by other
   specifications that might include some or all additional JWK members,
   if use cases arise where such different kinds of thumbprints would be
   useful. See Section 9.1 of RFC 7517 [JWK] for notes on some ways to
   cryptographically bind attributes to a key.

3.3. Order and Representation of Members in Hash Input

   The required members in the input to the hash function are ordered
   lexicographically by the Unicode code points of the member names.

   Characters in member names and member values MUST be represented
   without being escaped. This means that thumbprints of JWKs that
   require such characters are not defined by this specification. (This
   is not expected to limit the applicability of this specification, in
   practice, as the members of JWK representations are not expected to
   use any of these characters.) The characters specified as requiring
   escaping by Section 7 of [RFC7159] are quotation mark, reverse
   solidus (a.k.a. backslash), and the control characters U+0000 through
   U+001F.

   If the JWK key type uses members whose values are themselves JSON
   objects, then the members of those objects MUST likewise be
   lexicographically ordered. (As of the time of this writing, none are
   defined that do.)

   If the JWK key type uses members whose values are JSON numbers, and
   if those numbers are integers, then they MUST be represented as a
   JSON number as defined in Section 6 of [RFC7159] without including a
   fraction part or exponent part. For instance, the value "1.024e3"

   MUST be represented as "1024". This means that thumbprints of JWKs
   using numbers that are not integers are not defined by this
   specification. Also, as noted in "The I-JSON Message Format"
   [RFC7493], implementations cannot expect an integer whose absolute
   value is greater than 9007199254740991 (i.e., that is outside the
   range [-(2**53)+1, (2**53)-1]) to be treated as an exact value. (As
   of the time of this writing, none are defined that use JSON numbers.)

   See Section 4 for a discussion of further practical considerations
   pertaining to the representation of the hash input.

3.4. Selection of Hash Function

   A specific hash function must be chosen by an application to compute
   the hash value of the hash input. For example, SHA-256 [SHS] might
   be used as the hash function by the application. While SHA-256 is a
   good default choice at the time of this writing, the hash function of
   choice can be expected to change over time as the cryptographic
   landscape evolves.

   Note that in many cases, only the party that creates a key will need
   to know the hash function used. A typical usage is for the producer
   of the key to use the base64url-encoded JWK Thumbprint value as a
   "kid" (key ID) value. In this case, the consumer of the "kid" treats
   it as an opaque value that it uses to select the key.

   However, in some cases, multiple parties will be reproducing the JWK
   Thumbprint calculation and comparing the results. In these cases,
   the parties will need to know which hash function was used and use
   the same one.

3.5. JWK Thumbprints of Keys Not in JWK Format

   Note that a key need not be in JWK format to create a JWK Thumbprint
   of it. The only prerequisites are that the JWK representation of the
   key be defined and the party creating the JWK Thumbprint be in
   possession of the necessary key material. These are sufficient to
   create the hash input from the JWK representation of the key, as
   described in Section 3.3.

4. Practical JSON and Unicode Considerations

   Implementations will almost certainly use functionality provided by
   the platform's JSON support when parsing the JWK and emitting the
   JSON object used as the hash input. As a practical consideration,
   future JWK member names and values should be avoided for which
   different platforms or libraries might emit different
   representations. As of the time of this writing, all defined JWK

   member names and values use only printable ASCII characters, which
   should not exhibit this problem. Note however, that JSON.stringify()
   cannot be counted on to lexicographically sort the members of JSON
   objects, so while it could be used to emit some kinds of member
   values, different code is likely to be needed to perform the sorting.

   In particular, while the operation of lexicographically ordering
   member names by their Unicode code points is well defined, different
   platform sort functions may produce different results for non-ASCII
   characters, in ways that may not be obvious to developers. If
   writers of future specifications defining new JWK key type values
   choose to restrict themselves to printable ASCII member names and
   values (which are for machine and not human consumption anyway), some
   future interoperability problems might be avoided.

   However, if new JWK members are defined that use non-ASCII member
   names or values, their definitions should specify the exact Unicode
   code point sequences used to represent them. This is particularly
   important in cases in which Unicode normalization could result in the
   transformation of one set of code points into another under any
   circumstances.

   Use of escaped characters in JWKs for which JWK Thumbprints will be
   computed should be avoided. Use of escaped characters in the hash
   input JWKs derived from these original JWKs is prohibited.

   There is a natural representation to use for numeric values that are
   integers. However, this specification does not attempt to define a
   standard representation for numbers that are not integers or that
   contain an exponent component. This is not expected to be a problem
   in practice, as the required members of JWK representations are
   expected to use only numbers that are integers.

   Use of number representations containing fraction or exponent parts
   in JWKs for which JWK Thumbprints will be computed should be avoided.

   All of these practical considerations are really an instance of Jon
   Postel's principle: "Be liberal in what you accept, and conservative
   in what you send."

5. Relationship to Digests of X.509 Values

   JWK Thumbprint values are computed on the JWK members required to
   represent a key, rather than all members of a JWK that the key is
   represented in. Thus, they are more analogous to applications that
   use digests of X.509 Subject Public Key Info (SPKI) values, which are
   defined in Section 4.1.2.7 of [RFC5280], than to applications that
   use digests of complete certificate values, as the "x5t" (X.509

certificate SHA-1 thumbprint) [JWS] value defined for X.509
certificate objects does. While logically equivalent to a digest of
the SPKI representation of the key, a JWK Thumbprint is computed over
a JSON representation of that key, rather than over an ASN.1
representation of it.

6. IANA Considerations

   This specification adds to the instructions for the Designated
   Experts of the following IANA registries, all of which are in the
   "JSON Object Signing and Encryption (JOSE)" registry [IANA.JOSE]:

   o JSON Web Key Types
   o JSON Web Key Elliptic Curve
   o JSON Web Key Parameters

   IANA has added a link to this specification in the Reference sections
   of these registries.

   For these registries, because of the practical JSON and Unicode
   considerations described in Section 4, the Designated Experts must
   either:

   (a) require that JWK member names and values being registered use
   only printable ASCII characters excluding double quote ('"') and
   backslash ('\') (the Unicode characters with code points U+0021,
   U+0023 through U+005B, and U+005D through U+007E), or

   (b) if new JWK members or values are defined that use other code
   points, require that their definitions specify the exact Unicode code
   point sequences used to represent them. Furthermore, proposed
   registrations that use Unicode code points that can only be
   represented in JSON strings as escaped characters must not be
   accepted.

7. Security Considerations

   The JSON Security Considerations and Unicode Comparison Security
   Considerations described in Sections 10.12 and 10.13 of "JSON Web
   Signature (JWS)" [JWS] also apply to this specification.

   Also, as described in Section 4, some implementations may produce
   incorrect results if esoteric or escaped characters are used in the
   member names. The security implications of this appear to be limited
   for JWK Thumbprints of public keys, because while it may result in
   implementations failing to identify the intended key, it should not
   leak information. The information in a public key is already public
   in nature, by definition.

A hash of a symmetric key has the potential to leak information about
the key value. Thus, the JWK Thumbprint of a symmetric key should
typically be concealed from parties not in possession of the
symmetric key, unless in the application context, the cryptographic
hash used, such as SHA-256, is known to provide sufficient protection
against disclosure of the key value.

A JWK Thumbprint will only uniquely identify a particular key if a
single unambiguous JWK representation for that key is defined and
used when computing the JWK Thumbprint. (Such representations are
defined for all the key types defined in "JSON Web Algorithms (JWA)"
[JWA].) For example, if an RSA key were to use "e":"AAEAAQ"
(representing [0, 1, 0, 1]) rather than the specified correct
representation of "e":"AQAB" (representing [1, 0, 1]), then a
different thumbprint value would be produced for what could be
effectively the same key, at least for implementations that are lax
in validating the JWK values that they accept. Thus, JWK Thumbprint
values can only be relied upon to be unique for a given key if the
implementation also validates that the correct representation of the
key is used.

Even more insidious is that an attacker may supply a key that is a
transformation of a legal key in order to have it appear to be a
different key. For instance, if a legitimate RSA key uses a modulus
value N and an attacker supplies a key with modulus 3*N, the modified
key would still work about 1/3 of the time, but would appear to be a
different key. Thus, while thumbprint values are valuable for
identifying legitimate keys, comparing thumbprint values is not a
reliable means of excluding (blacklisting) the use of particular keys
(or transformations thereof).

8. References

8.1. Normative References

   [IANA.JOSE] IANA, "JSON Object Signing and Encryption (JOSE)",
               <http://www.iana.org/assignments/jose>.

   [JWA]   Jones, M., "JSON Web Algorithms (JWA)", RFC 7518,
               DOI 10.17487/RFC7518, May 2015,
               <http://www.rfc-editor.org/info/rfc7518>.

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

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

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

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

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

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

8.2. Informative References

   [RFC5280]   Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
                 Housley, R., and W. Polk, "Internet X.509 Public Key
                 Infrastructure Certificate and Certificate Revocation
                 List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May
                 2008, <http://www.rfc-editor.org/info/rfc5280>.

   [RFC7493]   Bray, T., Ed., "The I-JSON Message Format", RFC 7493,
                 DOI 10.17487/RFC7493, March 2015,
                 <http://www.rfc-editor.org/info/rfc7493>.

Acknowledgements

Authors' Addresses

   Michael B. Jones
   Microsoft

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

   Nat Sakimura
   Nomura Research Institute

   Email: n-sakimura@nri.co.jp
   URI: http://nat.sakimura.org/