Internet Engineering Task Force (IETF)
Request for Comments: 7797
Updates: 7519
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft

February 2016

JSON Web Signature (JWS) Unencoded Payload Option

Abstract

JSON Web Signature (JWS) represents the payload of a JWS as a base64url-encoded value and uses this value in the JWS Signature computation. While this enables arbitrary payloads to be integrity protected, some have described use cases in which the base64url encoding is unnecessary and/or an impediment to adoption, especially when the payload is large and/or detached. This specification defines a means of accommodating these use cases by defining an option to change the JWS Signing Input computation to not base64url-encode the payload. This option is intended to broaden the set of use cases for which the use of JWS is a good fit.

This specification updates RFC 7519 by stating that JSON Web Tokens (JWTs) MUST NOT use the unencoded payload option defined by this 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/rfc7797.

Table of Contents

1. Introduction

The "JSON Web Signature (JWS)" [JWS] specification defines the JWS Signing Input as the input to the digital signature or Message Authentication Code (MAC) computation, with the value ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)). While this works well in practice for many use cases, including those accommodating arbitrary payload values, other use cases have been described in which base64url-encoding the payload is unnecessary and/or an impediment to adoption, particularly when the payload is large and/or detached.

This specification introduces a new JWS Header Parameter value that generalizes the JWS Signing Input computation in a manner that makes base64url-encoding the payload selectable and optional. The primary set of use cases where this enhancement may be helpful are those in which the payload may be very large and where means are already in place to enable the payload to be communicated between the parties without modifications. Appendix F of [JWS] describes how to represent JWSs with detached content, which would typically be used for these use cases.

The advantages of not having to base64url-encode a large payload are that allocation of the additional storage to hold the base64url-encoded form is avoided and the base64url-encoding computation never has to be performed. In summary, this option can help avoid unnecessary copying and transformations of the potentially large payload, resulting in sometimes significant space and time improvements for deployments.

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

This specification uses the same terminology as the "JSON Web Signature" [JWS] and "JSON Web Algorithms" [JWA] specifications.

3. The "b64" Header Parameter

This Header Parameter modifies the JWS Payload representation and the JWS Signing Input computation in the following way:

b64
: The "b64" (base64url-encode payload) Header Parameter determines whether the payload is represented in the JWS and the JWS Signing Input as ASCII(BASE64URL(JWS Payload)) or as the JWS Payload value itself with no encoding performed. When the "b64" value is "false", the payload is represented simply as the JWS Payload value; otherwise, it is represented as ASCII(BASE64URL(JWS Payload)). The "b64" value is a JSON boolean, with a default value of "true". When used, this Header Parameter MUST be integrity protected; therefore, it MUST occur only within the JWS Protected Header. Use of this Header Parameter is OPTIONAL. If the JWS has multiple signatures and/or MACs, the "b64" Header Parameter value MUST be the same for all of them. Note that unless the payload is detached, many payload values would cause errors parsing the resulting JWSs, as described in Section 5.

The following table shows the JWS Signing Input computation, depending upon the value of this parameter:

"b64"	JWS Signing Input Formula
true	ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload))
false	ASCII(BASE64URL(UTF8(JWS Protected Header)) || '.') || JWS Payload

4. Examples

This section gives examples of JWSs showing the difference that using the "b64" Header Parameter makes. The examples all use the JWS Payload value [36, 46, 48, 50]. This octet sequence represents the ASCII characters "$.02"; its base64url-encoded representation is "JC4wMg".

The following table shows a set of Header Parameter values without using a false "b64" Header Parameter value and a set using it, with the resulting JWS Signing Input values represented as ASCII characters:

JWS Protected Header	JWS Signing Input Value
{"alg":"HS256"}	eyJhbGciOiJIUzI1NiJ9.JC4wMg
{"alg":"HS256","b64":false, "crit":["b64"]}	eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2U sImNyaXQiOlsiYjY0Il19.$.02

These examples use the Hash-based Message Authentication Code (HMAC) key from Appendix A.1 of [JWS], which is represented below as a JSON Web Key [JWK] (with line breaks within values for display purposes only):

```
{
 "kty":"oct",
 "k":"AyM1SysPpbyDfgZld3umj1qzKObwVMkoqQ-EstJQLr_T-1qS0gZH75
      aKtMN3Yj0iPS4hcgUuTwjAzZr1Z9CAow"
}
```

The rest of this section shows complete representations for the two JWSs above.

4.1. Example with Header Parameters {"alg":"HS256"}

The complete JWS representation for this example using the JWS Compact Serialization and a non-detached payload (with line breaks for display purposes only) is:

```
eyJhbGciOiJIUzI1NiJ9
.
JC4wMg
.
5mvfOroL-g7HyqJoozehmsaqmvTYGEq5jTI1gVvoEoQ
```

Note that this JWS uses only features defined by [JWS] and does not use the new "b64" Header Parameter. It is the "control" so that differences when it is used can be easily seen.

The equivalent representation for this example using the flattened JWS JSON Serialization is:

```
{
 "protected":
  "eyJhbGciOiJIUzI1NiJ9",
 "payload":
  "JC4wMg",
 "signature":
  "5mvfOroL-g7HyqJoozehmsaqmvTYGEq5jTI1gVvoEoQ"
}
```

4.2. Example with Header Parameters {"alg":"HS256","b64":false,"crit":["b64"]}

The complete JWS representation for this example using the JWS Compact Serialization and a detached payload (with line breaks for display purposes only) is:

```
eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19
.
.
A5dxf2s96_n5FLueVuW1Z_vh161FwXZC4YLPff6dmDY
```

Note that the payload "$.02" cannot be represented in this JWS in its unencoded form because it contains a period ('.') character, which would cause parsing problems. This JWS is therefore shown with a detached payload.

The complete JWS representation for this example using the flattened JWS JSON Serialization and a non-detached payload is:

```
{
 "protected":
  "eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19",
 "payload":
  "$.02",
 "signature":
  "A5dxf2s96_n5FLueVuW1Z_vh161FwXZC4YLPff6dmDY"
}
```

If using a detached payload with the JWS JSON Serialization, the "payload" element would be omitted.

5. Unencoded Payload Content Restrictions

When the "b64" value is "false", different restrictions on the payload contents apply, depending upon the circumstances, as described in this section. The restrictions prevent the use of payload values that would cause errors parsing the resulting JWSs.

Note that because the character sets that can be used for unencoded non-detached payloads differ between the two serializations, some JWSs using a "b64" value of "false" cannot be syntactically converted between the JWS JSON Serialization and the JWS Compact Serialization. See Section 8 for security considerations on using unencoded payloads.

5.1. Unencoded Detached Payload

Appendix F of [JWS] describes how to represent JWSs with detached content. A detached payload can contain any octet sequence representable by the application. The payload value will not cause problems parsing the JWS, since it is not represented as part of the JWS. If an application uses a content encoding when representing the payload, then it MUST specify whether the signature or MAC is performed over the content-encoded representation or over the unencoded content.

5.2. Unencoded JWS Compact Serialization Payload

When using the JWS Compact Serialization, unencoded non-detached payloads using period ('.') characters would cause parsing errors; such payloads MUST NOT be used with the JWS Compact Serialization. Similarly, if a JWS using the JWS Compact Serialization and a non-detached payload is to be transmitted in a context that requires URL-safe characters, then the application MUST ensure that the payload contains only the URL-safe characters 'a'-'z', 'A'-'Z', '0'-'9', dash ('-'), underscore ('_'), and tilde ('~'). The payload value is the ASCII representation of the characters in the payload string. The ASCII space character and all printable ASCII characters other than period ('.') (those characters in the ranges %x20-2D and %x2F-7E) MAY be included in a non-detached payload using the JWS Compact Serialization, provided that the application can transmit the resulting JWS without modification.

No meaning or special semantics are attached to any characters in the payload. For instance, the percent ('%') character represents itself and is not used by JWS objects for percent-encoding [RFC3986]. Applications, of course, are free to utilize content-encoding rules of their choosing, provided that the encoded representations utilize only allowed payload characters.

5.3. Unencoded JWS JSON Serialization Payload

When using the JWS JSON Serialization, unencoded non-detached payloads must consist of the octets of the UTF-8 encoding of a sequence of Unicode code points that are representable in a JSON string. The payload value is determined after performing any JSON string escape processing, per Section 8.3 of RFC 7159 [RFC7159], and then UTF-8-encoding the resulting Unicode code points. This means, for instance, that these payloads represented as JSON strings are equivalent ("$.02", "\u0024.02"). Unassigned Unicode code point values MUST NOT be used to represent the payload.

6. Using "crit" with "b64"

The "crit" Header Parameter MUST be included with "b64" in its set of values when using the "b64" Header Parameter to cause implementations not implementing "b64" to reject the JWS (instead of it being misinterpreted).

7. Intended Use by Applications

Application profiles should specify whether "b64" with a "false" value is to be used by the application in each application context or not, with it then being consistently applied in each application context. For instance, an application that uses detached payloads might specify that "b64" with a "false" value always be used. It is NOT RECOMMENDED that this parameter value be dynamically varied with different payloads in the same application context.

While it is legal to use "b64" with a "true" value, it is RECOMMENDED that "b64" simply be omitted in this case, since it would be selecting the behavior already specified in [JWS].

For interoperability reasons, JSON Web Tokens [JWT] MUST NOT use "b64" with a "false" value.

8. Security Considerations

[JWS] base64url-encodes the JWS Payload to restrict the set of characters used to represent it so that the representation does not contain characters used for delimiters in JWS representations. Those delimiters are the period ('.') character for the JWS Compact Serialization and the double-quote ('"') character for the JWS JSON Serialization. When the "b64" (base64url-encode payload) value is "false", these properties are lost. It then becomes the responsibility of the application to ensure that payloads only contain characters that will not cause parsing problems for the serialization used, as described in Section 5. The application also incurs the responsibility to ensure that the payload will not be modified during transmission.

Note that if a JWS were to be created with a "b64" value of "false" without including the "crit" Header Parameter with "b64" in its set of values and it were to be received by an implementation not supporting the "b64" Header Parameter, then the signature or MAC would still verify but the recipient would believe that the intended JWS Payload value is the base64url decoding of the payload value received, rather than the payload value received itself. For example, if the payload value received were "NDA1", an implementation not supporting this extension would interpret the intended payload as

being the base64url decoding of this value, which is "405". Requiring the use of the "crit" Header Parameter with "b64" in the set of values prevents this misinterpretation.

9. IANA Considerations

9.1. JSON Web Signature and Encryption Header Parameter Registration

This specification registers the "b64" Header Parameter defined in Section 3 in the IANA "JSON Web Signature and Encryption Header Parameters" registry [IANA.JOSE] established by [JWS].

9.1.1. Registry Contents

- Header Parameter Name: "b64"
- Header Parameter Description: Base64url-Encode Payload
- Header Parameter Usage Location(s): JWS
- Change Controller: IESG
- Specification Document(s): Section 3 of RFC 7797

10. References

10.1. 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>.

[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>.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

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

[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>.

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

[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>.

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

10.2. Informative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

Acknowledgements

Anders Rundgren, Richard Barnes, Phillip Hallam-Baker, Jim Schaad, Matt Miller, Martin Thomson, and others have all made the case for being able to use a representation of the payload that is not base64url encoded in contexts in which it safe to do so.

Thanks to Sergey Beryozkin, Stephen Farrell, Benjamin Kaduk, James Manger, Kathleen Moriarty, Axel Nennker, Anders Rundgren, Nat Sakimura, Jim Schaad, Robert Sparks, and Matias Woloski for their reviews of the specification, and thanks to Vladimir Dzhuvinov for verifying the examples.

Author’s Address

Michael B. Jones
Microsoft

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