JOSE Working Group
Internet-Draft
Updates: 7519 (if approved)
Intended status: Standards Track
Expires: May 14, 2016

M. Jones
Microsoft

November 11, 2015

JWS Unencoded Payload Option
draft-ietf-jose-jws-signing-input-options-04

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 prohibiting the use of this option in JSON Web Tokens (JWTs).

Status of this Memo

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

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

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

This Internet-Draft will expire on May 14, 2016.

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 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 specification 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)" [JWS] and "JSON Web Algorithms (JWA)" [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}	eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2V 9.$.02

These examples use the HMAC key from Appendix A.1 of [JWS], which is represented below as a JWK [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}

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

```
eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2V9
.
.
GsyM6AQJbQHY8aQKCbZSPJHzMRWo3HKIlcDuXof7nqs
```

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":
  "eyJhbGciOiJIUzI1NiIsImI2NCI6ZmFsc2V9",
 "payload":
  "$.02",
 "signature":
  "GsyM6AQJbQHY8aQKCbZSPJHzMRWo3HKIlcDuXof7nqs"
}
```

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 7 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. Intended Use by Applications

It is intended that application profiles specify up front 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.

JSON Web Tokens (JWTs) [JWT] MUST NOT use "b64" with a "false" value.

7. Security Considerations

[JWS] base64url-encodes the JWS Payload to restrict the character set used to represent it to characters that are distinct from the delimiters that separate it from other JWS fields. 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, and that the payload will not be modified during transmission.

There is no security problem if a JWS correctly created using "b64" with a "false" value is received by an implementation not supporting the "b64" Header Parameter, since the signature will fail to verify and the JWS will therefore be rejected. Likewise, there is no security problem if a JWS is created by an implementation not supporting this extension and received by an implementation supporting it, since the JWS will not use the extension, meaning that the security considerations are the same as for implementations supporting only the functionality specified in [JWS].

The only case in which care may need to be taken is when a JWS is possibly being received by a JWS implementation not supporting this

extension and the party producing the JWS might be an attacker that intentionally creates a JWS with a "b64" value of "false" but signs it as if its value was "true". In this case, a JWS implementation not supporting this extension will accept the JWS but treat it as if the payload was encoded, rather than unencoded.

While this confused case may seem like a problem that could matter, in fact, if the recipient trusts the creator of a JWS based on its signature and therefore accepts and acts upon the content in the JWS, yet the creator is an attacker, the recipient has much bigger problems than confusion between whether the payload value is encoded or unencoded. In this case, the attacker can sign any payload whatsoever and have the recipient accept it -- including payloads designed to cause harm to the recipient. If the trust established by verifying the signer's key does not actually establish that the creator is a trusted party, then this case in which JWS libraries supporting and not supporting the extension may respectively interpret the attacker's payload as being encoded or unencoded is the least of the application's worries. This can only happen when the trust placed in the creator of the JWS is unfounded.

The other relevant perspective on this case is that if an application specifies that JWSs used by it are to be created using "b64" with a "false" value, per Section 6, then correct implementations of that application must use a JWS implementation that implements this extension. Thus, even malicious JWS creators intentionally incorrectly signing JWSs as if the "b64" value was "false" cannot confuse the application as to whether the payload is encoded or unencoded, since any incorrectly signed JWSs will be rejected due to the bad signature. Only if the application is incorrectly implemented and doesn't support this extension when the application requires its use, is a problem possible. But then again, if the application is incorrectly implemented, it likely has bigger problems than confusion about whether the payload is encoded or not.

Only if the application dynamically switches between "false" and "true" values for "b64" (something NOT RECOMMENDED in Section 6), would it be necessary for the application to require the use of "crit" with a value of "["b64"]" in such application contexts. That would cause even incorrectly implemented application clients that do not support the extension to nonetheless reject content signed using "b64" with a "false" value.

8. IANA Considerations

8.1. JWS and JWE 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].

8.1.1. 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 [[this specification]]

9. References

9.1. 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/>.

9.2. 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>.

Appendix A. 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, James Manger, Axel Nennker, Anders Rundgren, Nat Sakimura, Jim Schaad, and Matias Woloski for their reviews of the specification.

Appendix B. Document History

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

-04

- Corrected a typo in the JWS JSON Serialization example.
- Added to the security considerations, including adding a statement about when "crit" should be used.
- Addressed the document shepherd comments.

-03

- Allowed the ASCII space character and all printable ASCII characters other than period ('.') in non-detached unencoded

payloads using the JWS Compact Serialization.

- Updated the abstract to say that that the spec updates RFC 7519.
- Removed unused references.
- Changed the change controller to IESG.

-02

- Required that "b64" be integrity protected.
- Stated that if the JWS has multiple signatures and/or MACs, the "b64" Header Parameter value MUST be the same for all of them.
- Stated that if applications use content encoding, they MUST specify whether the encoded or unencoded payload is used as the JWS Payload value.
- Reorganized the Unencoded Payload Content Restrictions section.
- Added an "updates" clause for RFC 7519 because this specification prohibits JWTs from using "b64":false.

-01

- Removed the "sph" (secure protected header) Header Parameter.
- Changed the title to "JWS Unencoded Payload Option".
- Added the section "Unencoded Payload Content Restrictions".
- Added an example using the JWS JSON Serialization.

-00

- Created the -00 JOSE working group draft from draft-jones-jose-jws-signing-input-options-00 with no normative changes.

Author's Address

Michael B. Jones
Microsoft

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