Workgroup: Network Working Group
Internet-Draft:
draft-waite-jws-multi-payload-00
Published: 10 July 2023
Intended Status: Informational
Expires: 11 January 2024
Authors: D. Waite, Ping Identity
J. Miller, Ping Identity
M. Jones, individual

JSON Web Signatures (JWS) Multiple Payload Option

Abstract

The JOSE set of standards established JSON-based container formats for signatures over a content payload using established algorithms.

Newer algorithms are emerging which allow for additional operations on content, such as a party (other than the signer) choosing not to disclose some of the integrity-protected content. However, these algorithms often support granularity at the individual message level, creating a need to define a way to support expressing multiple content payloads as part of a single message.

This document defines a new operational mode for JSON Web Signatures that operates on a protected header and multiple binary content payloads to provide the expressivity needed for this class of algorithm. It also describes how multiple content payloads can be expressed in a manner compatible with pre-existing algorithms, albeit without the operational capabilities of newer algorithms.

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 https://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 11 January 2024.

Copyright Notice

Table of Contents

1. Introduction

The JOSE specifications are very widely deployed and well supported, enabling use of cryptographic primitives with a JSON representation. JWTs [RFC7519] are one of the most common representations for identity and access claims. For instance, they are used by the OpenID Connect and Secure Telephony Identity Revisited (STIR) standards.

JWTs are also used by W3C's Verifiable Credentials and are used in many Decentralized Identity systems, where they may represent rich identity claims about a subject as an issued statement, which may be presented at some point in the future to another party for verification, without active participation by the original issuer.

With these new use cases, there is an increased focus on adopting privacy-protecting cryptographic primitives. The privacy-protection focus is largely in two areas: allowing a party to reduce the amount of information from the original document which is presented to a third party, and reducing correlation from the cryptographic algorithms when presenting a single issued statement multiple times.

One commonality across these algorithms is that they either require or are computationally simplified by delineating information items into multiple content payloads (whether the algorithms refer to them as messages, attributes, or slots), which are bound together into a single cryptographic object. They then define transformations in order to modify these individual components to release, omit, or express statements on the value of those components.

This specification defines an operational mode for algorithms which are multi-payload aware, as well as JSON and compact expressions for multiple payloads. It also defines how multiple payloads can be processed by algorithms which do not support the above transformations, and how multiple payloads might be used pre-existing JWS implementations based on such algorithms.

2. Conventions and Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification will use the following conventions to describe expressions for transforms and serializations:

1. A function is a transform or other operation which takes inputs and outputs a value will be described using a capitalized function name, with parameters folling in parenthesis as a comma-separated list, such as FUNCTION(INPUT1, INPUT2). Functions may nest if the input to one function is the output of another.

2. A constant value is indicated with a capitalized name and no parameterized list, such as CONSTANT

3. Constants which are single 7-bit ASCII codes are indicated by the value, surrounded by single quotes, such as '~'

4. Multiple values are concatenated into a single serialized form using two vertical line or "pipe" characters, e.g. A || B

5. Optionality is indicated by surrounding an expression with square brackets, such as A || [B || C]

The following additional conventions are used for expressing serialization of variable-length lists

1. The values in a list, if needed, is indicated using a 1-based numerical suffix, such as Payload 1

2. A loop is indicated by double square brackets, using a suffix of n, such as Payload 1 || [['~' || Payload n]]

This specification uses the same terminology as the "JSON Web Signature" [JWS] and "JSON Web Algorithms" [JWA] specifications, as well as the BASE64URL(OCTETS), UTF8(STRING), and ASCII(STRING) encoding conventions (here referred to as functions.)

3. Multiple Payload Aware Algorithms

Algorithms which support multiple payloads do not operate on the JWS Signing Input. They instead expect to receive information directly and individually as octets:

*The octets from UTF-8 encoding the protected header

*An ordered list of content payloads, each expressed as octets. Note that a single payload may be a zero length octet string, or may be omitted.

> NOTE: Need to decide if zero length and nil are worth representing as distinct, partially as compact representation will not be able to distinguish them.

Neither the protected header nor the content payloads are input in their JSON or compact serialized form, and thus are not expected to have a BASE64URL encoding applied when input for signature or validation.

Algorithms which are not aware of multiple payloads are instead expected to operate on the JWS signing input form, described below.

4. Multi Payload Serializations

4.1. JSON Serialization

For the JWS JSON serialization, multiple payloads are expressed via the new "payloads" member, which is an array where each entry is either a base64url-encoded content payload value or the JSON value null. Implementations MUST verify the "payload" member is absent when "payloads" is present.

4.2. Compact Serialization

For the JWS Compact serialization, multiple payloads are expressed by base64url-encoding each, then concatenating them into a single textual value with the tilde '~' character. This value is then expressed in lieu of a single base64-url encoded payload.

```
BASE64URL(UTF8(JWS Protected Header))
|| '.' || BASE64URL(JWS Payload 1) ||
[[ '~' || BASE64URL(JWS Payload n) ]]
|| '.' || BASE64URL(JWS Signature)
```

For example, if the protected header coincidentally base64url-encoded to "HEADER", the three payloads base64url-encoded to "PAYLOAD1", "PAYLOAD2", and "PAYLOAD3", and the signature to "SIGN", the compact serialization would be:

```
HEADER.PAYLOAD1~PAYLOAD2~PAYLOAD3.SIGN
```

JWS Compact serialization represents omitted payloads as zero length payloads, and both base64url-encode to a zero length character sequence. If the second payload value had been omitted, the representation would have been:

```
HEADER.PAYLOAD1~~PAYLOAD3.SIGN
```

5. JWS Signing Input

For algorithms which are not multiple payload aware, they are expected to continue to operate on a JWS signing input. When Multiple payloads are used, the JWS signing input is:

```
BASE64URL(JWS Protected Header)
|| '.' || BASE64URL(JWS Payload 1) ||
[[ '~' || BASE64URL(JWS Payload n) ]]
```

6. The "mp" Header Parameter

This Header Parameter indicates the signature is protecting multiple content payloads.

The value true modifies the representation in JSON and compact encodings, as well as the JWS signing input, to to the rules above.

Multi-payload aware algorithms cannot operate on JWS signing input, and MUST be assumed to be operating as if "mp" was specified as true. A "mp" header of false is not legal in this scenario, and it is RECOMMENDED that the "mp" header not be specified.

Applications which do not specify multi-payload behavior can be assumed to be operating in a mode where "mp" is false. Applications MAY either indicate this value be specified explicitly, or be assumed by context.

7. Interactions with Unencoded Payload Option

[RFC7797] specifies the unencoded payload option, which allows for payloads that can be expressed without base64url-encoding to skip the payload transformation, altering transforms as well as the JWS signed input. This is done via the "b64" protected header being true. The payload in such a case can include both the base64url alphabet as well as the tilde character ~.

As the unencoded payload option describes how to encoded multiple payloads, the "b64" protected header does not have an effect on multi-payload processing. That said, the two headers have compatible payloads and JWS signing input, by noting such an unencoded payload input is a *combined payload serialization* of the multi-payload input, defined as:

```
BASE64URL(JWS Payload 1) ||
[[ '~' || BASE64URL(JWS Payload n) ]]
```

7.1. Compatibility mode with implementations without multi-payload support

The unencoded payload option can be used in concert with multi-payload support when using algorithms which are not multi-payload aware, and communicating with compact serialization. This provides compatibility with JWS implementations without multi-payload support, which will fall back to interpreting the payload as a combined payload serialization. For such implementations, another layer of the application would be responsible for decomposing and interpreting the combined payload.

When operating in compatibility mode, the protected header should indicate:

- *a non multi-payload-aware algorithm

*a "mp" header of true, indicating that multiple payloads for JWS implmentations which support such a feature

*a "b64" header of false, indicating that the payload is not encoded

*a "crit" header including "b64" but not including "mp".

As this is the only valid combination of "mp" with "b64" as false, a multi-payload aware JWS implementation SHOULD consider that they satisfy the "crit" requirement for "b64", even if they otherwise do not support unencoded payloads.

Due to the difference in JSON serialization between the payloads value defined for multi-payload support and the payload value expected by the unencoded payload option, you MUST NOT use JSON serialization for transmission when operating with this combination of header parameters.

8. Indicating Multi-payload is required

When not using the compatibility mode described above, the JWS MUST use existing mechanisms to indicate the requirements of the message, and MUST NOT rely on side effects such as base64url decoding errors to prevent consumption by incompatible implementations.

As such, the following two mechanisms are described to explicitly limit compatibility:

1. A multi-payload-aware algorithm MUST only be supported by a multi-payload compatible JWS implementation. Implementations which do not understand multiple payloads will fail when they encounter an algorithm they do not support

2. When an "mp" header of true is used with an algorithm that is not multiple payload aware, a "crit" header including "mp" MUST be supplied.

9. Detached payload content

Appendix F of [JWS] describes how to represent JWSs with detached content, by applications omitting the payload in the transmitted serialization, and having the application reconsistitute the payload member to do integrity verification.

The steps to detach all payloads from a multi-payload JWS are similar, with the caveat that the JSON serialization would now have the payloads key omitted in such a scenario.

It is RECOMMENDED that applications describe when and how the detached content is to be used, taking particular caution around confusion that could result if only *some* of the payloads have been detached.

10. Security Considerations

TODO Security

11. IANA Considerations

11.1. JSON Web Signature and Encryption Header Parameter Registration

This specification registers the "mp" Header Parameter defined in Section Section 6 of this specification in the IANA " JSON Web Signature and Encryption Header Parameters" registry established by [JWS]

11.1.1. Registry Contents

*Header Parameter Name: "mp"

*Header Parameter Description: Multiple Payload Encoding o Header Parameter Usage Location(s): JWS o Change Controller: IESG o Specification Document(s): TBD

12. Normative References

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

13. Informative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Acknowledgments

TODO acknowledge.

Authors' Addresses

David Waite

Ping Identity

Email: dwaite+jose@pingidentity.com

Jeremie Miller
Ping Identity

Email: jmiller@pingidentity.com

Michael B. Jones
individual

Email: michael_b_jones@hotmail.com
URI: https://self-issued.info/