Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 7, 2016

A. Rundgren, Ed.
WebPKI.org

November 4, 2015

Predictable Serialization for JSON Tools
draft-rundgren-predictable-serialization-for-json-00

Abstract

This specification outlines an optional characteristic of JSON tools like parsers, serving two entirely different purposes: 1) Making information-rich JSON messages more human-readable by honoring the originator's conventions. 2) Facilitating simple "Signed JSON" schemes without necessarily needing specific signature text-processing software. Finally, there is a section containing recommendations for interoperability with systems based on EcmaScript V6 (AKA JavaScript).

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 7, 2016.

Table of Contents

1. Introduction

There is currently a strong trend moving from XML, EDI, ASN.1, and plain-text formats to JSON [RFC7159]. Although obviously working, JSON's unspecified of ordering of properties as well as the lack of a canonical form, sometimes make the transition rather painful.

The sample below displays the problems in a nutshell. Assume the following JSON message is parsed:

```
{
  "device": "Pump2",
  "value": 0.000000000000000001
}
```

After serialization a fully JSON-compliant output may look like:

```
{
  "value": 1e-18,
  "device": "Pump2"
}
```

Note: Whitespace was added for brevity.

If a JSON object contains dozens of properties the ability for a human to follow a message with respect to its specification (which presumably lists properties in a "logical" order), becomes considerably harder if the properties are serialized in an arbitrary order. In addition, changing the representation of numbers also contributes to confusion. Computers however, do not care.

While limitations in JSON-data for human consumption may only be considered a "nuisance", adding a signature property to a JSON object is infeasible unless there is some kind of predictable representation of data. This is one of the reasons why JSON Web Signature (JWS) [RFC7515] specifies that data to be signed must be Base64URL-encoded which though unfortunately makes JWS-signed messages unreadable by humans.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Predictable Serialization

To cope with the mentioned drawbacks, this specification introduces a simple predictable serialization scheme, preferably implemented directly in JSON parsers.

Note: This is not an attempt to change the JSON language in any way; it is only about how it is processed!

2.1. Ordering of Properties

The original property order MUST be honored during parsing of JSON objects to support a subsequent serialization phase. Duplicate or empty properties MUST be rejected.

2.2. Element Handling

In addition to preserving property order, this specification implies specific handling of JSON language elements, described in the succeeding sub-sections.

2.2.1. Whitespace Processing

All whitespace before and after JSON values and structural characters MUST be removed during serialization.

2.2.2. String Normalization

Quoted strings including properties MUST be normalized in a way which is close to the de-facto standard for JSON parsers which is:

- JSON '\/' escape sequences MUST be honored on input within quoted strings but be treated as a "degenerate" equivalents to '/' by rewriting them.

- Unicode [UNICODE] escape sequences ('\uhhhh') within quoted strings MUST be adjusted as follows: If the Unicode value falls within the ASCII [RFC20] control character range (0x00 - 0x1f), it MUST be rewritten in lower-case hexadecimal notation unless it is one of the pre-defined JSON escapes ('\n' etc.) because the latter have precedence. If the Unicode value is outside of the ASCII control character range, it MUST be replaced by the corresponding Unicode character with the exception of '"' and '\' which always MUST be escaped as well.

2.2.3. Number Representation

The textual representation of numbers MUST be preserved during parsing and serialization. That is, if numbers like 3.50 and -0 are encountered during a parsing process, they MUST be serialized as 3.50 and -0 respectively although 3.5 and 0 would be the most natural outcome.

2.2.4. Other Element Types

No particular action needs to be taken for the remaining JSON language elements.

3. Signed JSON Objects

The following non-normative section shows the principles for creating and verifying in-object signatures built on top of the predictable serialization concept.

3.1. Creating a Signed JSON object

Assume there is a JSON object like the following:

```
  {
    "property-1": ...,
    "property-2": ...,
    ...
    "property-n": ...
  }
```

A with this specification compliant serialization would then return:

```
  {"property-1":...,"property-2":...,..."property-n":...}
```

This string may after conversion to UTF-8 [RFC3629] be signed using any suitable algorithm like HS256 [RFC7518] or RS256 [RFC7518]. Using a bare-bones signature scheme the resulting JSON object could look like the following:

```
  {
    "property-1": ...,
    "property-2": ...,
    ...
    "property-n": ...,
    "signature":"LmTlQxXB3LgZrNLmhOfMaCnDizczC_RfQ6Kx8iNwfFA"
  }
```

The actual signature value would typically be Base64URL-encoded [RFC4648].

Note: The placement of the "signature" property with respect to the other properties (1-n) is insignificant.

Note: Signed data may very well be "pretty-printed" since whitespace is excluded by the serialization process.

3.2. Verifying a Signed JSON Object

The signed object created in the previous section could be verified by performing the following steps:

1. Parse the JSON object

2. Read and decode the value of the "signature" property

3. Remove the "signature" property from the JSON object

4. Serialize the JSON object which should generate exactly the same result as in the preceding section

5. Apply an algorithm-dependent signature verification method using the signature key, the UTF-8 representation of the serialization result from step #4, and the data read in step #2 as input arguments

3.3. Interoperability with EcmaScript V6

Since ECMAScript [ECMA-262] due its availability in Internet browsers represents the largest base of JSON-tools, it seems likely that "Signed JSON" will also be used in such environments. This is indeed possible but there are some constraints that need to be catered for if interoperability with this specification is desired:

- Property names MUST NOT be expressed as integer values like "1" because EmcaScript does not honor creation order for such items as described in section 9.1.12 of the EcmaScript V6 specification.

- Serialization of floating-point numbers is described in section 7.1.12.1 of the EcmaScript V6 specification. However, since this serialization scheme does not guarantee the correctness of the least significant digit, the following workaround is REQUIRED for maintaining interoperability between different EcmaScript implementations:

```
var aValue = 0.000000000000000001;
var myObject = {};
myObject.device = 'Pump2';
myObject.value =
    parseFloat((Math.abs(aValue) < 2.22507385850721E-308 ?
        0 : aValue).toPrecision(15));

// Serialize object to a JSON string
var jsonString = JSON.stringify(myObject);

// This string can now be signed and the value be
// added to the object itself (not shown here)
```

The test with 2.22507385850721E-308 is for dealing with underflow and 15 digits of precision at the same time.

Non-EcmaScript systems targeting EcmaScript environments MUST (of course) apply the measures specified above as well. An externally created signed object could for example be supplied as in-line EcmaScript in an HTML document like below:

```
var inObjectSignedData =
  {
    "device": "Pump2",
    "value": 1e-18,
    "signature": "LmTlQxXB3LgZrNLmhOfMaCnDizczC_RfQ6Kx8iNwfFA"
  };
```

Note: Whitespace can be used to make code more readable without affecting signatures.

Note: Quotes around property names are actually redundant if you (as in the example), stick to names that are syntactically compatible with the EcmaScript language.

4. Acknowledgements

During the initial design of the JSON Cleartext Signature (JCS) [JCS] scheme which was the "inspiration" for this specification, highly appreciated feedback was provided by Manu Sporny, Jim Klo, Jeffrey Walton, David Chadwick, Jim Schaad, David Waite, Douglas Crockford, Arne Riiber, Sergey Beryozkin, and Brian Campbell.

A special thank goes to James Manger who helped weeding out bugs in both the specification and in the reference code.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

This specification does (according to the author), not reduce or add vulnerabilities to JSON processing. Bugs in serializing software can though (of course) potentially expose sensitive data to attackers, activate protected APIs, or incorrectly validate signatures.

7. References

7.1. Normative References

[RFC20] Cerf, V., "ASCII format for Network Interchange", 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>.

[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/>.

7.2. Informative References

[ECMA-262] Ecma International, "ECMAScript Language Specification Edition 6", June 2015, <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.

[JCS] Rundgren, A., "JSON Cleartext Signature (JCS)", January 2015, <https://cyberphone.github.io/openkeystore/resources/docs/jcs.html>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC7515] 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>.

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

Author's Address

Anders Rundgren (editor)
WebPKI.org
14 Ave. Du General Leclerc
Perols 34470
France

Phone: +33 644 75 23 31
Email: anders.rundgren.net@gmail.com