CBOR
Internet-Draft
Intended status: Standards Track
Expires: 28 November 2025

L. Lundblade
Security Theory LLC

27 May 2025

CBOR Serialization and Determinism
draft-lundblade-cbor-serialization-01

Abstract

This document updates and clarifies CBOR Serialization and Deterministic Encoding as defined in [RFC8949]. It also provides background explanations that were not included in the original specification.

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 28 November 2025.

Table of Contents

1. Introduction

This document provides a complete definition of both Preferred Serialization and CBOR Deterministic Encoding Requirements (CDER) such that the reader does not need to refer to their definitions in [RFC8949].

The overwhelming purpose of this document is clarity and ease for the CBOR ecosystem on the subject of serialization and determinism. Aside from one small change, this restatement of the requirements doesn't change anything in [RFC8949]. No new concepts or terminology is introduced.

The small change is to Preferred Serialization. The conditional "preference" for deterministic length encoding in Section 4.1 of [RFC8949] is promoted to an unconditional requirement by this document. This change is considered reasonably compatible with the extant CBOR ecosystem. Since the publication of [RFC8949], a period of five years, the CBOR community largely assumed deterministic length encoding was a requirement of Preferred Serialization. It is better to make this minor change than to create a third serialization concept that would compound the complexity and confusion in this part of the CBOR ecosystem.

2. Information Model, Data Model and Serialization

To understand CBOR serialization and determinism, it's helpful to distinguish between the general concepts of an information model, a data model, and serialization.

	Information Model	Data Model	Serialization
Abstraction Level	Top level; conceptual	Realization of information in data structures and data types	Actual bytes encoded for transmission
Example	The temperature of something	A floating-point number representing the temperature	Encoded CBOR of a floating-point number
Standards		CDDL	CBOR
Implementation Representation		API Input to CBOR encoder library, output from CBOR decoder library	Encoded CBOR in memory or for transmission

Table 1

CBOR doesn’t provide facilities for information models. They are mentioned here for completeness and to provide some context.

CBOR defines a palette of basic types that are the usual integers, floating-point numbers, strings, arrays, maps and other. Extended types may be constructed from these basic types. These basic and extended types are used to construct the data model of a CBOR protocol. While not required, [RFC8610] may be used to describe the data model of a protocol. The types in the data model are serialized per [RFC8949] to create encoded CBOR.

CBOR allows certain data types to be serialized in multiple ways to facilitate easier implementation in constrained environments. For example, indefinite-length encoding enables strings, arrays, and maps to be streamed without knowing their length upfront.

Crucially, CBOR allows — and even expects — that some implementations will not support all serialization variants. In contrast, JSON permits variations (e.g., representing 1 as 1, 1.0, or 0.1e1), but expects all parsers to handle them. That is, the variation in JSON is for human readability, not to facilitate easier implementation in some environments.

Since CBOR does not require implementations to support every serialization variant, defining a common serialization format is highly beneficial for those that don’t need specialized encoding. This is the role of preferred serialization. It mandates a specific variant for each data type when multiple options exist.

3. Preferred Serialization

The requirements in the next two sections replace the definition of Preferred Serialization in [RFC8949].

They are restated in normative form to be more clear and so they can be formally referenced by the restatement of Section 4.

To claim support for Preferred Serialization, everything output by an encoder MUST meet all these requirements. In particular this disallows using tag 2 and 3 for values in the range of -(2^64) to 2^64 - 1.

As mentioned in Section 1 there is one change relative to the definition of Preferred Serialization in [RFC8949].

3.1. Encoder Requirements

1. Shortest-form encoding of the argument MUST be used for all major types. The shortest form encoding for any argument that is not a floating point value is:

   * 0 to 23 and -1 to -24 MUST be encoded in the same byte as the major type.

   * 24 to 255 and -25 to -256 MUST be encoded only with an additional byte (ai = 0x18).

   * 256 to 65535 and -257 to -65536 MUST be encoded only with an additional two bytes (ai = 0x19).

   * 65536 to 4294967295 and -65537 to -4294967296 MUST be encoded only with an additional four bytes (ai = 0x1a).

2. If maps or arrays are emitted, they MUST use definite-length encoding (never indefinite-length).

3. If text or byte strings are emitted, they MUST use definite-length encoding (never indefinite-length).

4. If floating-point numbers are emitted, the following apply:

   * The length of the argument indicates half (binary16, ai = 0x19), single (binary32, ai = 0x1a) and double (binary64, ai = 0x1b) precision encoding. If multiple of these encodings preserve the precision of the value to be encoded, only the shortest form of these MUST be emitted. That is, encoders MUST support half-precision and single-precision floating point. Positive and negative infinity and zero MUST be represented in half-precision floating point.

   * NaNs, and thus NaN payloads MUST be supported.

     As with all floating point numbers, NaNs with payloads MUST be reduced to the shortest of double, single or half precision that preserves the NaN payload. The reduction is performed by removing the rightmost N bits of the payload, where N is the difference in the number of bits in the significand (mantissa) between the original format and the reduced format. The reduction is performed only (preserves the value only) if all the rightmost bits removed are zero.

5. If big numbers (tags 2 and 3) are supported, the Preferred Serialization requirements decribed in Section 5 MUST be implemented.

3.2. Decoder Requirements

1. Decoders MUST accept shortest-form encoded arguments.

2. If arrays or maps are supported, definite-length arrays or maps MUST be accepted.

3. If text or byte strings are supported, definite-length text or byte strings MUST be accepted.

4. If floating-point numbers are supported, the following apply:

   * Half-precision values MUST be accepted.

   * Double- and single-precision values SHOULD be accepted; leaving these out is only foreseen for decoders that need to work in exceptionally constrained environments.

   * If double-precision values are accepted, single-precision values MUST be accepted.

   * NaNs, and thus NaN payloads, MUST be accepted.

5. If big numbers (tags 2 and 3) are supported, the Preferred Serialization requirements decribed in Section 5 MUST be implemented.

3.3. When to use Preferred Serialization

It is recommended that Preferred Serialization be used unless an application has special needs.

It is usually implementations in constrained environments that have special needs. For example, indefinite-length encoding is useful to send a lot of data from a device that has insufficient memory to store the data to be sent.

4. CBOR Deterministic Encoding Requirements

The requirements in the next two sections replace the definition of CDER from [RFC8949]:

There are no differences between these requirements and those of [RFC8949]. This restatement is only for the sake of clarity. ([RFC8949] allowed indefinite-length encoding for preferred serialization but not for CDER; that is why there is a change to preferred serialization in this document but not to CDER).

To claim support for CDER, everything output by an encoder MUST meet all these requirements. In particular this disallows using tag 2 and 3 for values in the range of -(2^64) to 2^64 - 1.

4.1. Encoder Requirements

1. Preferred Serialization defined in Section 3.1 MUST be used.

2. If a map is emitted, the keys in it MUST be sorted in the bytewise lexicographic order of their deterministic encodings.

4.2. Decoder Requirements

1. Decoders MUST meet the decoder requirements for Section 3.2. That is, deterministic encoding imposes no requirements over and above the requirements for decoding Preferred Serialization.

4.3. When to use Deterministic Serialization

Most applications do not require deterministic encoding— even those that use signing or hashing to authenticate or protect the integrity of data. For example, the payload of a COSE_Sign message does not need to be encoded deterministically, because it is transmitted along with the message. The recipient receives the exact same bytes that were signed.

Deterministic encoding becomes important when the data being protected is NOT transmitted in the form needed for authenticity or integrity checks— typically when that form is derived from other data. This can happen for reasons such as data size, privacy concerns, or other constraints.

The only difference between preferred and non-deterministic serialization is map key sorting. Sorting can be prohibitively expensive in very constrained environments. However, in many systems, sorting maps is not costly, and deterministic encoding can be used by default. Deterministically encoded data is always decodable, even by receivers that do not specifically support deterministic encoding. It can also be helpful for debugging protocols.

5. Clarified Big Number Requirements

5.1. Big Number Requirements

This text replaces Section 3.4.3 of [RFC8949].

Tag numbers 2 and 3 are used to represent bignums, which encode arbitrary-precision integers.

The content of a bignum tag MUST be a byte string, interpreted as an unsigned integer n in network byte order (big-endian).

* For tag 2 (positive bignum), the value is n.

* For tag 3 (negative bignum), the value is -1 - n.

Decoders MUST accept and ignore leading zeros in the byte string of a bignum. Decoders MUST also accept an empty byte string and treat it as representing the value zero.

CBOR defines both Preferred and Non-Preferred Serializations for bignums. The Preferred Serialization of bignums is also deterministic; therefore, no additional requirements are needed for deterministic encoding beyond those of Preferred Serialization.

5.1.1. Preferred Serialization

In Preferred Serialization, the bignum number space is unified with CBOR major types 0 and 1. This means that any value that can be represented using major type 0 (unsigned integers) or 1 (negative integers) MUST NOT be encoded as a bignum.

For example, the value 1 MUST be encoded as the single-byte 0x01, and MUST NOT be encoded as a bignum (e.g., 0xc24101).

Additionally, bignums in Preferred Serialization MUST NOT be encoded with leading zeros.

5.1.2. Non-Preferred Serialization

In Non-Preferred Serialization, the unification of the bignum number space with major types 0 and 1 does not apply. Values that can be represented using major types 0 or 1 MAY instead be encoded as bignums.

For example, the value 1 MAY be encoded as 0xc24101.

5.1.3. Implementation Guidance

Implementations operating in environments that support 128-bit integers SHOULD implement tags 2 and 3 to encode and decode those values.

5.2. Background Discussion on Preferred Serialization of Big Numbers

The requirement for Preferred Serialization of big numbers is atypical. The following subsections explain why this is the case and why this design choice was made. This background is intended to help clarify the distinction between serialization and data models, as the inclusion of this particular requirement in Preferred Serialization blurs that line.

5.2.1. Preferred Serialization Background

As mentioned in Section 2, CBOR was intentionally designed to allow variation in serialization to support implementation in constrained environments. All requirements associated with Preferred Serialization and the CBOR Deterministic Encoding Rules (CDER) — except those concerning big numbers — exist to eliminate this variation in deployments that do not require it. These requirements provide interoperability and determinism for the CBOR major types and don’t affect anything at the data model level.

For example, the Preferred Serialization requirement to use the shortest form of an argument eliminates variability in how arguments are encoded — variability that is sometimes intentionally made use of in constrained environments.

5.2.2. Data Model Background

Section 2 distinguishes serialization from data models. In CBOR data models, the major types (such as types 0 and 1 for integers) and tags (such as tags 2 and 3 for bignums) are distinct constructs. The way a particular value is serialized is separate from how it manifests in the data model level.

For instance, a floating-point number remains the same at the data model level regardless of whether it is serialized as a half-, single-, or double-precision float. The encoding is orthogonal to its meaning in the data model.

Similarly, major types 0 and 1 are conceptually separate from tags 2 and 3. Unifying these two representations — by requiring values encodable with major types 0 or 1 to be encoded that way instead of using tags 2 or 3 — amounts to a unification at the data model level. As such, this requirement is unlike other Preferred Serialization constraints, which are focused solely on serialization variability.

5.2.3. Rationale for Including Unification in Preferred Serialization

An alternate interpretation is that, since every value representable as a bignum within the range of major types 0 and 1 has an exact equivalent encoding using those major types, this is merely a matter of serialization choice. From this perspective, selecting one encoding form over the other could be seen as a serialization preference — even though it also implies unifying distinct data model types.

Another reason is that this is what [RFC8949] specified and this document seeks maximal compatibility with it.

Finally, including the unification rule in Preferred Serialization promotes consistent encoding of 128-bit integers. By incorporating this requirement, the specification encourages broader adoption and implementation of uniform 128-bit integer representations within CBOR.

6. Deterministic Encoding for Popular Tags

The definitions of the following tags in [RFC8610] allow variation in the data mode, thus it is useful to define a deterministic encoding for them should a particular deterministic protocol need one. The tags defined in [RFC8610] but not mentioned here have no variability in their data model.

6.1. Date Strings, Tag 0

TODO -- complete this work and remove this comment before publication

6.2. Epoch Date, Tag 1

6.2.1. Encoder Requirements

The integer form MUST be used unless one of the following applies: (1) the date is too far in the past or future to fit in a 64-bit integer of type 0 or 1, or (2) the date requires sub-second precision. In these cases, the floating-point form MUST be used instead.

6.2.2. Decoder Requirements

The decoder MUST decode both the integer and floating-point form.

6.3. Big Numbers, Tags 2 and 3

See Section 5.

6.4. Big Floats and Decimal Fractions, Tags 4 and 5

6.4.1. Encoder Requirements

The mantissa MUST be encoded in the preferred serialization form specified in Section 3.4.3 of RFC 8949.

The mantissa MUST NOT contain trailing zeros. For example, the decimal fraction with value 10 must be encoded with a mantissa of 1 and an exponent of 1. For big floats, the mantissa must not include any trailing zero bits if encoded as a type 0 or 1 integer, and no trailing zero bytes if encoded as a big number

6.4.2. Decoder Requirements

Both the integer and big number forms of the mantissa MUST be decoded.

7. General Protocol Considerations for Determinism

This is the section that covers what is know as ALDR in some discussions.

// RFC Editor: Please remove above sentence before publication

In addition to Section 4 and Section 6, there are considerations in the design of any deterministic protocol.

For a protocol to be deterministic, both the encoding (serialization) and data model (application) layer must be deterministic. While CDER ensures determinism at the encoding layer, requirements at the application layer may also be necessary.

Here's an example application layer specification:

> At the sender's convenience, the birth date MAY be sent either as an integer epoch date or string date. The receiver MUST decode both formats.

While this specification is interoperable, it lacks determinism. There is variability in the data model layer akin to variability in the CBOR encoding layer when CDER is not required.

To make this example application layer specification deterministic, specify one date format and prohibit the other.

A more interesting source of application layer variability comes from CBOR's variety of number types. For instance, the number 2 can be represented as an integer, float, big number, decimal fraction and other. Most protocols designs will just specify one number type to use, and that will give determinism, but here's an example specification that doesn't:

> At the sender's convenience, the fluid level measurement MAY be encoded as an integer or a floating-point number. This allows for minimal encoding size while supporting a large range. The receiver MUST be able to accept both integers and floating-point numbers for the measurement.

Again, this ensures interoperability but not determinism—identical fluid level measurements can be represented in more than one way. Determinism can be achieved by allowing only floating-point, though that doesn't minimize encoding size.

A better solution requires the fluid level always be encoded using the smallest representation for every particular value. For example, a fluid level of 2 is always encoding as an integer, never as a floating-point number. 2.000001 is always be encoded as a floating-point number so as to not lose precision. See the numeric reduction defined by dCBOR.

Although this is not strictly a CBOR issue, deterministic CBOR protocol designers should be mindful of variability in Unicode text, as some characters can be encoded in multiple ways.

While this is not an exhaustive list of application-layer considerations for deterministic CBOR protocols, it highlights the nature of variability in the data model layer and some sources of variability in the CBOR data model (i.e., in the application layer).

8. CDDL Support

TODO -- complete work and remove this comment

9. Security Considerations

The security considerations in Section 10 of [RFC8949] apply.

10. IANA Considerations

TODO -- complete work and remove this comment before publication

11. Normative References

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

Appendix A. Examples and Test Vectors

TODO -- complete work and remove this comment before publication

Author’s Address

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com