Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 11 February 2026

W. McNally
C. Allen
Blockchain Commons
C. Bormann
Universit辰t Bremen TZI
L. Lundblade
Security Theory LLC

10 August 2025

dCBOR: A Deterministic CBOR Application Profile
draft-mcnally-deterministic-cbor-13

Abstract

The purpose of determinism is to ensure that semantically equivalent data items are encoded into identical byte streams. CBOR (RFC 8949) defines "Deterministically Encoded CBOR" in its Section 4.2, but leaves some important choices up to the application developer. The CBOR Common Deterministic Encoding (CDE) Internet Draft builds on this by specifying a baseline for application profiles that wish to implement deterministic encoding with CBOR. The present document provides an application profile "dCBOR" that can be used to help achieve interoperable deterministic encoding based on CDE for a variety of applications wishing an even narrower and clearly defined set of choices.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mcnally-deterministic-cbor/.

Source for this draft and an issue tracker can be found at https://github.com/BlockchainCommons/WIPs-IETF-draft-deterministic-cbor.

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/.

Copyright Notice

Table of Contents

1. Introduction

CBOR [RFC8949] has many advantages over other data serialization formats. One of its strengths is specifications and guidelines for serializing data deterministically, such that multiple agents serializing the same data automatically achieve consensus on the exact byte-level form of that serialized data. This is particularly useful when data must be compared for semantic equivalence by comparing the hash of its contents.

Nonetheless, determinism is an opt-in feature of CBOR, and most existing CBOR codecs put the primary burden of correct deterministic serialization and validation of deterministic encoding during deserialization on the engineer. Furthermore, the specification leaves a number of important decisions around determinism up to the application developer. The CBOR Common Deterministic Encoding (CDE) Internet Draft [CDE] builds on the basic CBOR specification by providing a baseline for application profiles that wish to implement deterministic encoding with CBOR.

This document narrows CDE further into a set of requirements for the application profile "dCBOR". These requirements include but go beyond CDE, including requiring that dCBOR decoders validate that encoded CDE conforms to the requirements of this document.

1.1. Conventions and Definitions

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.

2. Application Profile

The dCBOR Application Profile specifies the use of Deterministic Encoding as defined in [CDE] and adds several exclusions and reductions specified in this section.

Just as CDE does not "fork" CBOR, the rules specified here do not "fork" CDE: A dCBOR implementation produces well-formed, deterministically encoded CDE according to [CDE], and existing CBOR or CDE decoders will therefore be able to decode it. Similarly, CBOR or CDE encoders will be able to produce valid dCBOR if handed dCBOR conforming data model level information from an application.

Note that the separation between standard CBOR or CDE processing and the processing required by the dCBOR application profile is a conceptual one: Both dCBOR processing and standard CDE/CBOR processing may be combined into a unified dCBOR/CDE/CBOR codec. The requirements in this document apply to encoding or decoding of dCBOR data, regardless of whether the codec is a unified dCBOR/CDE/CBOR codec operating in dCBOR-compliant modes, or a single-purpose dCBOR codec. Both of these are generically referred to as "dCBOR codecs" in this document.

This application profile is intended to be used in conjunction with an application, which typically will use a subset of CDE/CBOR, which in turn influences which subset of the application profile is used. As a result, this application profile places no direct requirement on what subset of CDE/CBOR is implemented. For instance, there is no requirement that dCBOR implementations support floating point numbers (or any other kind of non-basic integer type, such as arbitrary precision integers or complex numbers) when they are used with applications that do not use them. However, this document does place requirements on dCBOR implementations that support negative 64-bit integers and 64-bit or smaller floating point numbers.

2.1. Common Deterministic Encoding Conformance

dCBOR encoders:

1. MUST only emit CBOR conforming "CBOR Common Deterministic Encoding (CDE)" [CDE], including mandated preferred encoding of integers and floating point numbers and the lexicographic ordering of map keys.

dCBOR decoders:

2. MUST validate that encoded CBOR conforms to the requirements of [CDE].

2.2. Duplicate Map Keys

CBOR [RFC8949] defines maps with duplicate keys as invalid, but leaves how to handle such cases to the implementor (則2.2, 則3.1, 則5.4, 則5.6). [CDE] provides no additional mandates on this issue.

dCBOR encoders:

1. MUST NOT emit CBOR maps that contain duplicate keys.

dCBOR decoders:

2. MUST reject encoded maps with duplicate keys.

2.3. Numeric Reduction

The purpose of determinism is to ensure that semantically equivalent data items are encoded into identical byte streams. Numeric reduction ensures that semantically equal numeric values (e.g. 2 and 2.0) are encoded into identical byte streams (e.g. 0x02) by encoding "Integral floating point values" (floating point values with a zero fractional part) as integers when possible.

dCBOR implementations that support floating point numbers:

1. MUST check whether floating point values to be encoded have the numerically equal value in DCBOR_INT = [-2^63, 2^64-1]. If that is the case, it MUST be converted to that numerically equal integer value before encoding it. (Preferred encoding will then ensure the shortest length encoding is used.) If a floating point value has a non-zero fractional part, or an exponent that takes it out of DCBOR_INT, the original floating point value is used for encoding. (Specifically, conversion to a CBOR bignum is never considered.)

   This also means that the three representations of a zero number in CBOR (0, 0.0, -0.0 in diagnostic notation) are all reduced to the basic integer 0 (with preferred encoding 0x00).

| Note that numeric reduction means that some maps that are valid CDE/CBOR cannot be reduced to valid dCBOR maps, as numeric reduction can result in multiple entries with the same keys ("duplicate keys"). For example, the following is a valid CBOR/CDE map:
|
```
{
   10: "ten",
   10.0: "floating ten"
}
```
|
| Figure 1: Valid CBOR data item with numeric map keys (also valid CDE)
|
| Applying numeric reduction to this map would yield the invalid map:
|
```
{  / invalid: multiple entries with the same key /
   10: "ten",
   10: "floating ten"
}
```

```
|
|         Figure 2: Numeric reduction turns valid CBOR invalid
|
|  In general, dCBOR applications need to avoid maps that have
|  entries with keys that are semantically equivalent in dCBOR's
|  numeric model.
```

2. MUST reduce all encoded NaN values to the quiet NaN value having the half-width CBOR representation 0xf97e00.

dCBOR decoders that support floating point numbers:

3. MUST reject any encoded floating point values that are not encoded according to the above rules.

2.4. Simple Values

Only the three "simple" (major type 7) values false (0xf4), true (0xf5), and null (0xf6) and the floating point values are valid in dCBOR.

dCBOR encoders:

1. MUST NOT encode major type 7 values other than false, true, null, and the floating point values.

dCBOR decoders:

2. MUST reject any encoded major type 7 values other than false, true, null, and the floating point values.

2.5. Strings

dCBOR encoders:

1. MUST only emit text strings that are in Unicode Normalization Form C (NFC) [UNICODE-NORM].

dCBOR decoders:

1. MUST reject any encoded text strings that are not in NFC.

3. CDDL support, Declarative Tag

Similar to the CDDL [RFC8610] support in CDE [CDE], this specification adds two CDDL control operators that can be used to specify that the data items should be encoded in CBOR Common Deterministic Encoding (CDE), with the dCBOR application profile applied as well.

The control operators .dcbor and .dcborseq are exactly like .cde and .cdeseq except that they also require the encoded data item(s) to conform to the dCBOR application profile.

Tag 201 (Section 6) is defined in this specification as a way to declare its tag content to conform to the dCBOR application profile at the data model level. As a result, when this data item is encoded using CDE rules, the encoded result will conform to dCBOR also at the encoded data item level. (In conjunction with this semantics, tag 201 may also be employed as a boundary marker leading from an overall structure to specific application data items; see Section 3 of [GordianEnvelope] for an example for this usage.)

4. Implementation Status

This section is to be removed before publishing as an RFC.

(Boilerplate as per Section 2.1 of [RFC7942]:)

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

4.1. Swift

* Description: Single-purpose dCBOR reference implementation for Swift.
* Organization: Blockchain Commons
* Implementation Location: [BCSwiftDCBOR]
* Primary Maintainer: Wolf McNally
* Languages: Swift
* Coverage: Complete
* Testing: Unit tests
* Licensing: BSD-2-Clause-Patent

4.2. Rust

* Description: Single-purpose dCBOR reference implementation for Rust.
* Organization: Blockchain Commons
* Implementation Location: [BCRustDCBOR]
* Primary Maintainer: Wolf McNally
* Languages: Rust
* Coverage: Complete
* Testing: Unit tests
* Licensing: BSD-2-Clause-Patent

4.3. TypeScript

* Description: Single-purpose dCBOR reference implementation for TypeScript.
* Organization: Blockchain Commons
* Implementation Location: [BCTypescriptDCBOR]
* Primary Maintainer: Wolf McNally

* Languages: TypeScript (transpiles to JavaScript)

* Coverage: Complete

* Testing: Unit tests

* Licensing: BSD-2-Clause-Patent

4.4. Ruby

* Implementation Location: [cbor-dcbor]

* Primary Maintainer: Carsten Bormann

* Languages: Ruby

* Coverage: Complete specification; complemented by CBOR encoder/decoder and command line interface from [cbor-diag] and deterministic encoding from [cbor-deterministic]. Checking of dCBOR - exclusions not yet implemented.

* Testing: Also available at https://cbor.me

* Licensing: Apache-2.0

5. Security Considerations

This document inherits the security considerations of CBOR [RFC8949].

Vulnerabilities regarding dCBOR will revolve around whether an attacker can find value in producing semantically equivalent documents that are nonetheless serialized into non-identical byte streams. Such documents could be used to contain malicious payloads or exfiltrate sensitive data. The ability to create such documents could indicate the failure of a dCBOR decoder to correctly validate according to this document, or the failure of the developer to properly specify or implement application protocol requirements using dCBOR. Whether these possibilities present an identifiable attack surface is a question that developers should consider.

6. IANA Considerations

RFC Editor: please replace RFCXXXX with the RFC number of this RFC and remove this note.

IANA has registered the following CBOR tag in the "CBOR Tags" registry of [IANACBORTAGS]:

Tag	Data Item	Semantics	Reference
#201	(any)	enclosed dCBOR	[RFCXXXX]

Table 1: CBOR Tag for dCBOR

This document requests IANA to register the contents of Table 1 into the registry "CDDL Control Operators" of [IANACDDL]:

Name	Reference
.dcbor	[RFCXXXX]
.dcborseq	[RFCXXXX]

Table 2: CDDL Control Operators for dCBOR

7. Appendix A: dCBOR Numeric Test Vectors

The following tables provide common and edge-case numeric test vectors for dCBOR encoders and decoders, and are intended to exercise the requirements of this specification.

7.1. dCBOR Numeric Encodings

Value	dCBOR Encoding	Note
0	00	
1	01	
23	17	
24	1818	
255 (2^8 - 1)	18ff	
65535 (2^16 - 1)	19ffff	
65536 (2^16)	1a00010000	
4294967295 (2^32 - 1)	1affffffff	

4294967296 (2^32)	1b0000000100000000	
18446744073709551615 (2^64 - 1)	1bffffffffffffff	
-1	20	
-2	21	
-127 (-2^8 - 1)	387e	
-128 (-2^7)	387f	
-32768 (-2^16)	397fff	
-2147483648 (-2^31)	3a7fffffff	
-9223372036854775808 (-2^63)	3b7fffffffffffff	
1.5	f93e00	
2345678.25	fa4a0f2b39	
1.2	fb3ff3333333333333	
42.0	182a	Reduced.
2345678.0	1a0023cace	Reduced.
-2345678.0	3a0023cacd	Reduced.
-0.0	00	Reduced.
5.960464477539063e-08	f90001	Smallest half-precision subnormal.
1.401298464324817e-45	fa00000001	Smallest single subnormal.
5e-324	fb0000000000000001	Smallest double subnormal.
2.2250738585072014e-308	fb0010000000000000	Smallest double normal.

6.103515625e-05	f90400	Smallest half-precision normal.
65504.0	19ffe0	Reduced. Largest possible half-precision.
33554430.0	1a01fffffe	Reduced. Exponent 24 to test single exponent boundary.
-9223372036854774784.0	3b7fffffffffffbff	Reduced. Most negative double that converts to int64.
18446744073709550000.0	1bfffffffffffff800	Reduced. Largest double that can convert to uint64, almost UINT64_MAX.
18446744073709552000.0	fa5f800000	Just too large to convert to uint64, but converts to a single, just over UINT64_MAX.
-18446742974197924000.0	fadf7fffff	Large negative that converts to float, but too large for int64.
3.4028234663852886e+38	fa7f7fffff	Largest possible single.
3.402823466385289e+38	fb47effffe0000001	Slightly larger than largest possible single.
1.7976931348623157e+308	fb7feffffffffffff	Largest double.
Infinity (any size)	f97c00	Canonicalized.
-Infinity (any size)	f9fc00	Canonicalized.

NaN (any size, any payload)	f97e00	Canonicalized.

Table 3

7.2. Invalid dCBOR Encodings

These are valid CBOR encodings that MUST be rejected as invalid by a dCBOR-compliant decoder.

Value	CBOR Encoding	Reason for Rejection
12.0	f94a00	Can be reduced to 12.
1.5	fb3ff8000000000000	Not preferred encoding.
-9223372036854775809 (-2^63 - 1)	3b8000000000000000	65-bit negative integer value.
-18446744073709551616 (-2^64)	3bffffffffffffffff	65-bit negative integer value.
Infinity	fb7ff0000000000000	Not preferred encoding.
Infinity	fa7f800000	Not preferred encoding.
-Infinity	fbfff0000000000000	Not preferred encoding.
-Infinity	faff800000	Not preferred encoding.
NaN	fb7ff9100000000001	Not canonical NaN.
NaN	faffc00001	Not canonical NaN.
NaN	f97e01	Not canonical NaN.

Table 4

8. References

8.1. Normative References

[CDE] Bormann, C., "CBOR Common Deterministic Encoding (CDE)", Work in Progress, Internet-Draft, draft-ietf-cbor-cde-12, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-cde-12>.

[IANACBORTAGS]
IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANACDDL] IANA, "Concise Data Definition Language (CDDL)", <https://www.iana.org/assignments/cddl>.

[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/rfc/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/rfc/rfc8174>.

[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>.

[UNICODE-NORM]
"Unicode Normalization Forms", n.d., <https://unicode.org/reports/tr15/>.

8.2. Informative References

[BCRustDCBOR]
McNally, W., "Deterministic CBOR (dCBOR) for Rust.", n.d., <https://github.com/BlockchainCommons/bc-dcbor-rust>.

[BCSwiftDCBOR]
McNally, W., "Deterministic CBOR (dCBOR) for Swift.", n.d., <https://github.com/BlockchainCommons/BCSwiftDCBOR>.

[BCTypescriptDCBOR]
McNally, W., "Deterministic CBOR (dCBOR) for Typescript.", n.d., <https://github.com/BlockchainCommons/bc-dcbor-ts>.

[cbor-dcbor]
Bormann, C., "PoC of the McNally/Allen dCBOR application-level CBOR representation rules", n.d., <https://github.com/cabo/cbor-dcbor>.

[cbor-deterministic]
Bormann, C., "cbor-deterministic gem", n.d., <https://github.com/cabo/cbor-deterministic>.

[cbor-diag]
Bormann, C., "CBOR diagnostic utilities", n.d., <https://github.com/cabo/cbor-diag>.

[GordianEnvelope]
McNally, W. and C. Allen, "The Gordian Envelope Structured Data Format", Work in Progress, Internet-Draft, draft-mcnally-envelope-09, 29 March 2025, <https://datatracker.ietf.org/doc/html/draft-mcnally-envelope-09>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

Acknowledgments

The authors are grateful for the contributions of Joe Hildebrand and Anders Rundgren in the CBOR working group.

Authors' Addresses

Wolf McNally
Blockchain Commons
Email: wolf@wolfmcnally.com

Christopher Allen
Blockchain Commons
Email: christophera@lifewithalacrity.com

Carsten Bormann
Universit辰t Bremen TZI
Email: cabo@tzi.org

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