Workgroup: Network Working Group
Internet-Draft:
draft-mcnally-deterministic-cbor-07
Published: 9 January 2024
Intended Status: Experimental
Expires: 12 July 2024
Authors: W. McNally
Blockchain Commons
C. Allen
Blockchain Commons
C. Bormann
Universität Bremen TZI

dCBOR: A Deterministic CBOR Application Profile

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

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 12 July 2024.

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 (§2.2, §3.1, §5.4, §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. "65-bit" Negative Integers

dCBOR limits the range of integers to those that can be contained in common 64-bit programming language integer types, either as a signed (int64 or i64) or unsigned (uint64 or u64) integer. In other words, integer values in the range DCBOR_INT = $[-2^{63}, 2^{64}-1]$ are valid.

CBOR integers in the basic generic data model have an argument of up to 64 bits; whether the value is interpreted as non-negative or

negative then depends on the additional bit provided by whether it is encoded as a major type 0 or 1 value.

Many programming languages offer a separate type that covers the entire range of major type 0 (such as uint64 or u64), but do not offer a type that provides the full range of negative integers that can be encoded in CBOR major type 1. (If a two's-complement signed type were to be used to cover both ranges in full, it would need to have at least 65 bits.) We therefore use the name NEG_65 for the range of negative numbers that can be encoded in major type 1, but do not fit into int64, i.e., $[-2^{64}, -2^{63} - 1]$. Integer values in this range are invalid in dCBOR.

dCBOR encoders:

1. **MUST NOT** encode CBOR integer values in the range NEG_65.

dCBOR decoders:

2. **MUST** reject CBOR integer values in the range NEG_65.

(As always with CBOR, whether the value is interpreted as non-negative or negative depends on whether it is encoded as a major type 0 or 1 value.)

Specific applications will, of course, further restrict ranges of integers that are considered valid for the application, based on their position and semantics in the CBOR data item.

2.4. 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 as defined above. 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.

2.5. 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.

3. CDDL support

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.

For example, the normative comment in Section 3 of [GordianEnvelope]:

```
leaf = #6.24(bytes)  ; MUST be dCBOR
```

...can now be formalized as:

```
leaf = #6.24(bytes .dcbor any)
```

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.

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 1: CDDL Control
Operators for dCBOR

7. References

7.1. Normative References

[CDE] Bormann, C., "CBOR Common Deterministic Encoding (CDE)", Work in Progress, Internet-Draft, draft-ietf-cbor-cde-01, 8 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-cde-01>.

[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>.

7.2. 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-05, 20 August 2023, <https://datatracker.ietf.org/doc/html/draft-mcnally-envelope-05>.

[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, Laurence Lundblade, 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
Universität Bremen TZI

Email: cabo@tzi.org