Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 May 2026

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

1 November 2025

dCBOR: Deterministic CBOR
draft-mcnally-deterministic-cbor-14

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 present document specifies dCBOR, a set of narrowing rules for CBOR that can be used to help achieve interoperable deterministic encoding for a variety of applications desiring a narrow 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 3
  1.1. Conventions and Definitions 3
2. Narrowing Rules 3
  2.1. Definite Length Items 4
  2.2. Preferred Serialization 4
  2.3. Ordered Map Keys 5
  2.4. Duplicate Map Keys 5
  2.5. Numeric Reduction 6
  2.6. Simple Values 7
  2.7. Strings 7
3. CDDL support, Declarative Tag 8
4. Implementation Status 8
  4.1. Swift 8
  4.2. Rust 9
  4.3. TypeScript 9
  4.4. Ruby 10
5. Security Considerations 10
6. IANA Considerations 10
7. Appendix A: dCBOR Numeric Test Vectors 11
  7.1. dCBOR Numeric Encodings 11
  7.2. Invalid dCBOR Encodings 14
8. Appendix B: Design Principles 15
  8.1. Why Numeric Reduction? 17
  8.2. Why Not undefined? 18
  8.3. Why only a single NaN? 18

8.4. Why not other simple values? 19
8.5. Limiting Principles 20
8.6. Why not define an API? 20
9. References 20
9.1. Normative References 20
9.2. Informative References 21
Acknowledgments 22
Authors' Addresses 22

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.

This document narrows CBOR to a set of requirements called "dCBOR". These requirements include choices left open in CBOR, but also go beyond, including requiring that dCBOR decoders validate that encoded CBOR 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. Narrowing Rules

This section specifies the exclusions and reductions that dCBOR applies to CBOR.

The rules specified here do not "fork" CBOR: A dCBOR implementation produces well-formed, deterministically encoded CBOR according to [RFC8949], and existing CBOR decoders will therefore be able to decode it. Similarly, CBOR 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 processing and the processing required by the dCBOR rules is a conceptual one: Both dCBOR processing and standard CBOR processing may be combined into a unified dCBOR/CBOR codec. The requirements in this document apply to encoding or decoding of dCBOR data, regardless of whether the codec is a unified dCBOR/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.

dCBOR is intended to be used in conjunction with an application, which typically will use a subset of CBOR, which in turn influences which subset of dCBOR that is used. As a result, dCBOR places no direct requirement on what subset of 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. Definite Length Items

CBOR [RFC8949] allows both "definite-length" and "indefinite-length" items for byte strings, text strings, arrays, and maps (則3.2).

dCBOR encoders:

1. MUST only emit "definite-length" items for byte strings, text strings, arrays, and maps.

dCBOR decoders:

2. MUST reject any encoded "indefinite-length" items for byte strings, text strings, arrays, and maps.

2.2. Preferred Serialization

CBOR [RFC8949] allows multiple possible encodings for the same data item, and defines a "preferred serialization" in 則4.1 to be used for deterministic encoding.

dCBOR encoders:

1. MUST only emit "preferred serialization".

dCBOR decoders:

2. MUST validate that encoded CBOR conforms to "preferred serialization", and reject any encoded CBOR that does not conform.

2.3. Ordered Map Keys

The last bullet item of CBOR [RFC8949] 則4.2.1 defines a "bytewise lexicographic order" for map keys to be used for deterministic encoding.

dCBOR encoders:

1. MUST only emit CBOR maps with keys in "bytewise lexicographic order".

dCBOR decoders:

2. MUST validate that encoded CBOR maps have keys in "bytewise lexicographic order", and reject any encoded maps that do not conform.

2.4. 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).

dCBOR encoders:

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

dCBOR decoders:

2. MUST reject encoded maps with duplicate keys.

2.5. 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 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 map:
|
|
```
{
   10: "ten",
   10.0: "floating ten"
}
```
|
| Figure 1: Valid CBOR data item with numeric map keys
|
| 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.

2.6. 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.

2.7. Strings

CBOR [RFC8949] allows text strings to be any valid UTF-8 string (則3.1). However, Unicode character sequences can represent the same text string in different ways, leading to variability in the encoding of semantically equivalent data items. Unicode Normalization Form C (NFC) [UNICODE-NORM] is a commonly used normalization form that eliminates such variability.

dCBOR encoders:

1. MUST only emit text strings that are in NFC.

dCBOR decoders:

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

3. CDDL support, Declarative Tag

CDDL [RFC8610] is a widely used language for specifying CBOR data models. This specification adds two CDDL control operators that can be used to specify that the data items should be encoded in dCBOR.

The control operators .dcbor and .dcborseq are exactly like .cbor and .cborseq as defined in [RFC8610] except that they also require the encoded data item(s) to conform to dCBOR.

Tag 201 (Section 6) is defined in this specification as a way to declare its tag content to conform to dCBOR at the data model level and the encoded data item level. (In conjunction with these 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. Appendix B: Design Principles

This section is non-normative.

dCBOR has a single overriding goal: to facilitate _determinism_.

This means to ensure or facilitate, as much as possible, that semantically equivalent data items are encoded as identical byte streams.

In general, this means reducing or eliminating _variability_ in the encoding of data items. Variability arises where more than one valid encoding is possible for a given data item, and a protocol designer must make a choice as to which encoding to use. These choices can be arbitrary, and different protocol designers may make different arbitrary, and equally valid choices.

One of the most common examples of this arises with typed numeric values, where a numeric field must be pre-assigned a type (e.g., signed or unsigned integer of 8, 16, 32, or 64 bits, floating point of 16, 32, or 64 bits, etc.) CBOR's basic numeric data model is typed, and requires that numeric values be encoded according to their type. This is a cognitive burden on protocol designers, and a source of variability, since there may be several ways to encode a given numeric value depending on the type assigned to it. Many developers would prefer to encode numeric values without worrying about types, and let the encoding format handle the details, including ensuring deterministic encoding.

While dCBOR cannot automatically eliminate all variability in the design of deterministic protocols, it can provide a set of narrowing rules within its scope and level of abstraction that reduce the number of choices that protocol designers need to make.

dCBOR makes no claim that these are the _only_ or _best_ possible narrowing rules for deterministic encoding for every application. But dCBOR does provide a set of well-defined, easy-to-understand, and easy-to-implement rules that can be deployed as a package to facilitate deterministic encoding for a wide variety of applications. Making these choices at the dCBOR level reduces cognitive burden for protocol designers, and decreases the risk of interoperability problems between different implementations.

Variability Source	dCBOR Rule
Indefinite or definite length items	Only definite Length Items
Multiple possible encodings for same data item	Only preferred serialization
Different orders for map keys	Only ordered map Keys
Duplicate map keys	Duplicate Map Keys disallowed
Semantically equivalent numeric values (e.g., 0, 0.0, -0.0)	Only a single encoding for each distinct value
Choice of null or undefined	Only null
Simple values other than false, true, null	Only false, true, null
Nontrivial NaNs (sign, signaling, payloads)	Single NaN
Equivalent strings with multiple Unicode representations	Only NFC text strings

Table 5

The sections below explain the rationale for some of these choices.

8.1. Why Numeric Reduction?

The numeric model of [RFC8949] provides three kinds of basic numeric types: unsigned integers (Major Type 0), negative integers (Major Type 1), and floating point numbers (shares major Type 7 with Simple Values). Not all applications require floating point values, and those that do not are unaffected by the presence of floating point numbers in the CBOR model. However, the RFC introduces the possibility of variability in certain places. For example, 則3.4.2 defines Tag 1 as "Epoch-Based Date/Time":

> Tag number 1 contains a numerical value counting the number of seconds from 1970-01-01T00:00Z in UTC time to the represented point in civil time.

> The tag content MUST be an unsigned or negative integer (major types 0 and 1) or a floating-point number (major type 7 with additional information 25, 26, or 27). Other contained types are invalid.

An inhabitant of Tag 1, as long as it represents an integral number of seconds since the epoch, could therefore be encoded as an integer _or_ a floating point number. dCBOR's numeric reduction rule ensures that such values are always encoded as integers, eliminating variability in the encoding of such values.

But this raises a larger policy question for determinism: If two numeric values are semantically equal, should they be encoded identically? dCBOR answers "yes" to this question, and numeric reduction is the mechanism by which this is achieved. This choice answers the determinism question in a way that is simple to understand and implement, and that works well for the vast majority of applications. The serialization is still typed, but the burden of choosing types is reduced for protocol designers, who can simply specify numeric fields without worrying about the details of how those numbers will be encoded.

8.2. Why Not undefined?

How to represent an absent value is a perennial question in data modeling. In general it is useful to have a value that represents a placeholder for a position where a value _could_ be present but is not. This could be used in a map to indicate that a key is bound but has no value, or in an array to indicate that a value at a particular index is absent. There are other sorts of absence as well, such as the absence of a key in a map. dCBOR cannot address all of these different notions of absence, but can and does address the lack of semantic clarity around the choice between null and undefined by choosing null as the sole representation of a placeholder for an absent value. null is widely used in data modeling, and has a clear and unambiguous meaning. In contrast, undefined is less commonly used, and its meaning can be ambiguous. By choosing null, dCBOR provides a single clear way to represent absent values, reducing variability.

8.3. Why only a single NaN?

How to represent the result of a computation like 1.0 / 0.0 is another perennial question in data modeling. The [IEEE754] floating point standard answers this question with the concept of "Not a Number" (NaN): a special value that represents an unrepresentable or undefined numerical result. However, the standard also specifies several bit fields within the NaN representation that can vary,

including the sign bit, whether the NaN is "quiet" or "signaling", and a payload field. These formations are useful in certain computational contexts, but have no general meaning in data modeling.

The problem of NaN is complicated by the fact that IEEE 754 specifies that all NaN values compare as "not equal" to all other numeric values, including themselves. This means that comparing any two NaN values, including identical ones, will always yield "not equal". The deeper problem this raises is that if you want to know what data a NaN might carry in its payload, you have to go to extraordinary lengths to extract that information, since you cannot simply compare two NaN values to determine whether they are the same.

This not only raises deterministic variability issues (the array [1, NaN, 3] could be encoded in multiple ways depending on the NaN representation used), but also security issues as an attacker could use different NaN representations to exfiltrate data or hide malicious payloads, knowing that any comparison of NaN values will fail.

Given that NaN has utility in general data modeling, but its specification complexities raise both determinism and security issues, dCBOR chooses to simplify the situation by requiring that all NaN values be encoded as the single quiet NaN value having the half-width CBOR representation 0xf97e00.

8.4. Why not other simple values?

[RFC8949] Major Type 7 defines a space of 256 code points for "simple values", and 則3.3 defines four simple values and assigns them code points in the Major Type 7 space: false (20), true (21), null (22), and undefined (23). We have already discussed the choice of null over undefined. However, the remaining code points in this space are listed as either "unassigned" or "reserved" and delegates the registry of simple values to the IANA CBOR Simple Tags Registry [IANASIMPLEVALUES], which lists no assigned values other than those four.

The implication of this is that the semantics of these other simple values are officially undefined, and they cannot simply be used as application-defined values without risking interoperability issues. dCBOR therefore chooses to limit use of simple values to the three well-defined values false, true, and null, which are widely used in data modeling and have clear and unambiguous meanings.

8.5. Limiting Principles

A limiting principle of dCBOR is that it concerns itself with the most common data items used in CBOR applications. As a result, dCBOR does not place requirements on the encoding or decoding of CBOR data items that are less commonly used in practice, such as bignums, complex numbers, or other tagged data items. dCBOR implementations are not required to support these data items, but if they do, they must support them within the rules of dCBOR.

Tags provide a useful "escape hatch" for applications that need to use data items not covered by dCBOR. For example, dCBOR applications can freely use Tag 2 or Tag 3 to encode bignums, which contain byte strings, and on which dCBOR places no restrictions beyond those that apply to all byte strings (definite length only). Similarly, the rare applications that need to convey nontrivial NaN values can use Tag 80, 81, or 82 as defined in the IANA CBOR Tags Registery [IANACBORTAGS]. These tags use byte strings to encode arrays of fixed-length IEEE 754 floating point values in big-endian byte order.

8.6. Why not define an API?

Because dCBOR mandates strictness in both encoding and decoding, and because of mechanisms it introduces such as numeric reduction, the question arises as to whether this document should specify an API, or at least a set of best practices, for dCBOR codec APIs. The authors acknowledge that such guidance might be useful, but since the purpose of dCBOR is to provide a deterministic encoding format, and because APIs can vary widely between programming languages and environments, the authors have chosen to not widen the scope of this document. We direct the reader to the several existing dCBOR implementations for guidance on API design.

9. References

9.1. Normative References

[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>.

[IANASIMPLEVALUES]
IANA, "Concise Binary Object Representation (CBOR) Simple Values", <https://www.iana.org/assignments/cbor-simple-values>.

[IEEE754] "IEEE Standard for Floating-Point Arithmetic", n.d., <https://ieeexplore.ieee.org/document/8766229>.

[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/>.

9.2. 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-10, 30 September 2025, <https://datatracker.ietf.org/doc/html/draft-mcnally-envelope-10>.

[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, Rohan Mahy, 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