Blockchain Commons Website

Creating Open & Interoperable, Secure & Compassionate Digital Infrastructure

Blockchain Commons advocates for the creation of open, interoperable, secure & compassionate digital infrastructure to enable people to control their own digital destiny and to maintain their human dignity online.

Blockchain Commons works with developer communities to design, build, and maintain secure & compassionate decentralized architectures & tools for digital assets & digital identity based on responsible key management; based on our Gordian Principles of independence, privacy, resilience, and openness; and based on our Self-Sovereign Identity Principles. Our goal is to reclaim human dignity & authority in the digital world. We also strive to educate & grow the blockchain community through online courses and our work with legislators and regulators.

Blockchain Commons is proudly a “not-for-profit” social benefit corporation, domiciled in Wyoming but operating world-wide. We have a strong commitment to open source and a defensive patent strategy: anyone can use or improve our tools, and no one can take them away.
WHAT IS BLOCKCHAIN COMMONS?

- We are a community that brings together stakeholders to collaboratively build open & interoperable, secure & compassionate infrastructure.
- We design decentralized solutions where everyone wins.
- We are a neutral “not-for-profit” that enables people to control their own digital destiny.
The Envelope Structured Data Format

Abstract

The envelope protocol specifies a structured format for hierarchical binary data focused on the ability to transmit it in a privacy-focused way. Envelopes are designed to facilitate "smart documents" and have a number of unique features including: easy representation of a variety of semantic structures, a built-in Merkle-like digest tree, deterministic representation using CBOR, and the ability for the holder of a document to selectively encrypt or elide specific parts of a document without invalidating the document structure including the digest tree, or any cryptographic signatures that rely on it.

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."
The Envelope Structured Data Format

Abstract

The envelope protocol specifies a structured format for hierarchical binary data focused on the ability to transmit it in a privacy-focused way. Envelopes are designed to facilitate “smart documents” and have a number of unique features including: easy representation of a variety of semantic structures, a built-in Merkle-like digest tree, deterministic representation using CBOR, and the ability for the holder of a document to selectively encrypt or elide specific parts of a document without invalidating the document structure including the digest tree, or any cryptographic signatures that rely on it.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/BlockchainCommons/envelope-internet-draft.

Status of This Memo

Copyright Notice

1. Introduction

Gordian Envelope was designed with two key goals in mind: to be Structure-Ready, allowing for the reliable and interoperable storage of information; and to
ENVELOPES HOLD MANY KINDS OF THINGS
ENVELOPES CAN DO MANY THINGS
struct Envelope {
    let subject: Envelope
    let assertions: [Assertion]
}

struct Assertion {
    let predicate: Envelope
    let object: Envelope
}
struct Envelope {
    let subject: Envelope
    let assertions: [Assertion]
}

struct Assertion {
    let predicate: Envelope
    let object: Envelope
}

"Alice" [ "knows": "Bob" ]
struct Envelope {
    let subject: Envelope
    let assertions: [Assertion]
}

struct Assertion {
    let predicate: Envelope
    let object: Envelope
}

"Alice" [
    "knows": "Bob"
]
struct Envelope {
    let subject: Envelope
    let assertions: [Assertion]
}

struct Assertion {
    let predicate: Envelope
    let object: Envelope
}

"Alice" [
    "knows": "Bob"
]
"Alice" [
  "knows": "Bob"
]
"Alice" [ "knows": "Bob" ]
"Alice" [ 
  "knows": "Bob"
]

ELIDED [ 
  "knows": "Bob"
]

"Alice" [ 
  "knows": "Bob"
  "01": "P1"
]
"Alice" [ 
  "knows": "Bob"
]

"Alice" [ 
  ELIDED: "Bob"
]

"Alice" [ 
  "knows" [ 
    "02": "P2"
  ] : "Bob"
]
"Alice" [  
  "knows": "Bob"
]

"Alice" [  
  "knows": ELIDED
]

"Alice" [  
  "knows": "Bob" [  
    "03": "P3"
  ]
]

3 - OBJECT
"Alice" [ 
  "knows": "Bob"
]

"Alice" [ 
  ELIDED
]

"Alice" [ 
  "knows": "Bob"
  ] [ 
    "04": "P4"
  ]
"Alice" [ "knows": "Bob" ]

{  
  "Alice" [  
    "knows": "Bob"  
  ]  
} [  
  "O5": "P5"  
]
[Envelope("Alice"), Envelope("Bob"), Envelope("Carol"), Envelope("David"))]
USE CASES

A user can elide their content:

https://github.com/BlockchainCommons/Gordian/blob/master/Envelope/Use-Cases/Educational.md#2-danika-restricts-her-revelations-elision

A user can elide and then later reveal their content:

https://github.com/BlockchainCommons/Gordian/blob/master/Envelope/Use-Cases/Software.md#7-amira-reveals-her-identity-progressive-trust

User-based herd privacy:

dCBOR
CBOR

‣ BINARY
‣ CONCISE
‣ SELF-DESCRIBING
‣ CONSTRAINED ENVIRONMENTS
‣ PLATFORM/LANGUAGE AGNOSTIC
‣ STANDARDIZED
‣ DETERMINISTIC
dCBOR: Deterministic CBOR Implementation Practices

Abstract

CBOR 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. Nonetheless, determinism is an opt-in feature of the specification, and most existing CBOR codecs put the primary burden of correct deterministic serialization and validation of deterministic encoding during deserialization on the encoder. This document specifies a set of norms and practices for CBOR codec implementors intended to support deterministic CBOR ("dCBOR") at the codec API level.

Discussion Venues

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

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

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."
Gordian dCBOR: Deterministic CBOR Implementation Practices

Abstract
CBOR 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. Nonetheless, determinism is an opt-in feature of the specification, and most existing CBOR codecs put the primary burden of correct deterministic serialization and validation of deterministic encoding during deserialization on the engineer. This document specifies a set of norms and practices for CBOR codec implementors intended to support deterministic CBOR ("dCBOR") at the codec API level.

Discussion Venues
This note is to be removed before publishing as an RFC.
Source for this draft and an issue tracker can be found at https://github.com/BlockchainCommons/WIPs-IETF-draft-dcbor.

Status of This Memo
Copyright Notice
1. Introduction
3. Envelope Format Specification

This section is normative, and specifies the binary format of envelopes in terms of its CBOR components and their sequencing. The formal language used is the Concise Data Definition Language (CDDL) [RFC8610]. To be considered a well-formed envelope, a sequence of bytes MUST be well-formed deterministic CBOR [RFC8949] and MUST conform to the specifications in this section.

https://datatracker.ietf.org/doc/draft-mcnally-envelope/
3. Envelope Format Specification

This section is normative, and specifies the binary format of envelopes in terms of its CBOR components and their sequencing. The formal language used is the Concise Data Definition Language (CDDL) [RFC8610]. To be considered a well-formed envelope, a sequence of bytes MUST be well-formed deterministic CBOR [DCBOR-DRAFT] and MUST conform to the specifications in this section.

https://datatracker.ietf.org/doc/draft-mcnally-envelope/
THE POINT OF DETERMINISM

- Eliminate choices for how to serialize particular data.
- Where possible, the API enforces these standards.
- Where not possible, developers MUST specify how to serialize and how to validate on deserializing.
- Multiple agents serializing the same data should automatically achieve consensus on the exact form of that data.
4.2. Deterministically Encoded CBOR

Some protocols may want encoders to only emit CBOR in a particular deterministic format; those protocols might also have the decoders check that their input is in that deterministic format. Those protocols are free to define what they mean by a "deterministic format" and what encoders and decoders are expected to do. This section defines a set of restrictions that can serve as the base of such a deterministic format.
4.2.1. Core Deterministic Encoding Requirements

- Variable-length integers MUST be as short as possible.
- Floating-point values MUST use the shortest form that preserves the value.
- Indefinite-length arrays and maps MUST NOT be used.
- Map keys MUST be sorted in bytewise lexicographic order of their deterministic encodings.
4.2.2. Additional Deterministic Encoding Considerations

- Protocols MUST specify the circumstances under which a data item MUST or MUST NOT be tagged.

- Protocols allowing the use of BigNums ≥ $2^{64}$ (tags 2 and 3) MUST specify whether values $<2^{64}$ MUST use regular integer encodings.

- Protocols allowing the use of floating-point numbers must decide how to encode values like -0.0, NaN/Signalling NaN, subnormal values, etc.
WHAT DOES BLOCKCHAIN COMMONS SAY?

- Deterministic encoding is essential for cryptographic “smart documents” like Gordian Envelope.
- All of our existing CBOR specs are already deterministic encoding-compliant.
- Being opinionated is good.
- Enforcing opinionated best practices at the software/API level is even better.
HOW MANY EXISTING CBOR IMPLEMENTATIONS DIRECTLY SUPPORT DETERMINISTIC ENCODING AS A CORE VALUE?

404

(none found)
DETERMINISTIC CBOR

SO NOW WE’VE BUILT TWO OF THEM . . .

dCBOR Swift
https://github.com/BlockchainCommons/BCSwiftDCBOR

dCBOR Rust
https://crates.io/crates/dcbor
GOALS FOR dCBOR

- Make it easy to write and read deterministic CBOR (dCBOR) that complies with RFC-8949 §4.2.1. Core Deterministic Encoding Requirements.
- Be strict about what is written and read.
  - Make it hard to write non-compliant dCBOR.
  - Make it an error to read non-compliant dCBOR.
- Facilitate as much as possible the considerations in RFC-8949 §4.2.2. Additional Deterministic Encoding Considerations.
Encoding of Maps

- RFC-8949: Map keys MUST be sorted in bytewise lexicographic order of their deterministic encodings.
- dCBOR libraries provide a special-purpose `Map` structure that keeps key-value pairs in canonical sorted order as they are inserted or removed.
- Provides iteration through key-value pairs in canonical order.
- In some other ways they behave like a normal dictionary/map, but they are primarily intended for use during the serialization/deserialization process.
- Deserialization of out-of-order map keys is an error.
DETERMINISTIC CBOR

Encoding of Numeric Values

- All encoded numeric values use the shortest possible serialization.
  - Integers are 8, 16, 32, or 64 bits, floating point values 16, 32, or 64 bits.
- Floating point values with no fractional part are serialized as integers if possible.
  - This means that 0.0, -0.0, and 0 are all serialized exactly the same way.
  - Same semantics as JSON, JavaScript and Ruby.
- After deserialization, any numeric value can be extracted as a floating point value.
- Attempting to extract an integer from a numeric value with a fractional part is an error.
- Attempting to deserialize a dCBOR stream with any numeric values not in their canonical shortest form is an error.
- NaN is canonicalized to a single representation.
Provide protocols (Swift) and traits (Rust) to make structures CBOR-friendly.

- `CBORCodable` conformance adds serialization/deserialization to any type.
  - Many fundamental built-in types conform including integers, floating point values, strings, byte strings, arrays, booleans, and dates.
- `CBORTaggedCodable` adds a tag that is always written on serialization and expected on deserialization.
- No attempt has been made to make dCBOR compatible with either the `Codable` protocol (Swift) or the `SerDe` serialization framework (Rust).
  - A lot of work for little benefit, with many sharp edge/corner cases to deal with.
- If this is something you desire, we welcome PRs!
The deserialized `CBOR` type has `diagnostic()` and `hex()` methods.

Can be provided with `knownTags` argument that provides names for tags.

```
304( ; crypto-keypath
   {  
     1: [23, false, 23, true, 33, false]
   }
)
```

```
d9 0130       # tag(304) ; crypto-keypath
  a1         # map(1)
    01      # unsigned(1)
    86      # array(6)
      17    # unsigned(23)
      f4    # false
      17    # unsigned(23)
      f5    # true
      1821  # unsigned(33)
      f4    # false
```
DETERMINISTIC CBOR

Validations performed while decoding or extracting

```swift
public enum CBORError: LocalizedError, Equatable {
    /// Early end of data.
    case underrun
    /// Unsupported value in CBOR header.
    /// The case includes the encountered header as associated data.
    case badHeaderValue(associated: UInt8)
    /// A numeric value was encoded in non-canonical form.
    case nonCanonicalNumeric
    /// An invalidly-encoded UTF-8 string was encountered.
    case invalidString
    /// The decoded CBOR had extra data at the end.
    /// The case includes the number of unused bytes as associated data.
    case unusedData(Int)
    /// The decoded CBOR map has keys that are not in canonical order.
    case misorderedMapKey
    /// The decoded CBOR map has a duplicate key.
    case duplicateMapKey
    /// The numeric value could not be represented in the specified numeric type.
    case outOfRange
    /// The decoded value was not the expected type.
    case wrongType
    /// The decoded value did not have the expected tag.
    /// The case includes the expected tag and encountered tag as associated data.
    case wrongTag(expected: Tag, encountered: Tag)
    /// Invalid CBOR format. Frequently thrown by libraries depending on this one.
    case invalidFormat
}
```