Network Working Group
Internet-Draft
Intended status: Informational
Expires: 6 November 2026

R. Mahy

5 May 2026

Extended Diagnostic Notation (EDN) Use with Transport Layer Security (TLS) Presentation Language (PL) Objects
draft-mahy-cbor-edn-for-tls-00

Abstract

Extended Diagnostic Notation was designed as a superset of JSON to represent CBOR instance documents in human-readable format. This document describes how it can be used to represent instances encoded using the TLS Presentation Language.

About This Document

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

The latest revision of this draft can be found at https://rohanmahy.github.io/cbor-edn-for-tls/draft-mahy-cbor-edn-for-tls.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mahy-cbor-edn-for-tls/.

Source for this draft and an issue tracker can be found at https://github.com/rohanmahy/cbor-edn-for-tls.

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 6 November 2026.

Table of Contents

1. Introduction

Extended Diagnostic Notation (EDN) [I-D.ietf-cbor-edn-literals] is a text format, that was originally described to represent CBOR messages as diagnostic notation in [RFC7094] and then expanded in [RFC8949]. It is a superset of JSON (all valid JSON is valid EDN) that can natively represent binary content, non-finite floating point values, and additional data types (tags and simple values). Since then it has been used to represent YAML messages, especially those that contain binary content or tags.

This document defines a convention to represent instances of binary sequences--encoded using the TLS Presentation Language (TLS PL) defined in Section 3 of [RFC8446]--using EDN. TLS encoding is used in several other protocols, for example in the Messaging Layer Security (MLS) protocol [RFC9420] and the MIMI protocol [I-D.ietf-mimi-protocol].

TLS PL has a very limited set of patterns. It deals primarily with structs and enums. As a result, it is relatively straightforward to represent TLS encoded data if the TLS PL definitions are available.

2. 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.

3. Representation of TLS PL in EDN

EDN Representations of TLS-encoded instance data always start with the following comment line: # ~~~ tls followed by the struct name, and end with the following comment line: # ~~~. This allows a processor that knows TLS to convert an EDN document back into TLS-encoded binary data.

Instances of TLS-encoded structs are represented in EDN as maps with the keys equal to the names of the members.

If an enum appears that contains only the values false (0) and true (1), its values are represented in EDN as the simple values false and true.

Instances of other TLS-encoded enums are represented as their decimal integer values with an EDN end-of-line comment naming the enum name and the item name.

Instances of the following predefined TLS numeric types: uint16, uint32, uint64, plus any variable-length integers (widely used, for example in MLS Section 2.1.2 of [RFC9420]) are represented as unsigned decimal integers in EDN. Likewise a single instance of uint8 is represented as an unsigned decimal integer in EDN.

Vectors of uint8 (or the opaque type) are represented by single quoted strings in EDN. Likewise fixed-length arrays of uint8 are represented by single quoted strings in EDN. By default, they are presented as hex-encoded single-quoted strings. If such sequences consist primarily of printable characters

Vectors of any other type are represented by an array of that type.

For example given the FooBar struct defined below:

```
enum {
  false(0),
  true(1),
  (255)
} Bool;

enum {
  red(1),
  yellow(2),
  green(3)
  (255)
} Color;

struct {
  uint16 id;
  uint8[16] nonce;
  Bool active;
  Color traffic_light_color;
  uint32 divisible_by<V>;
  opaque reason;
} FooBar;
```

an instance of that struct could be represented as follows in EDN:

```
# ~~~ tls FooBar
{
  "id": 16798,
  "nonce": h'f6bafb33a535d1fd05bef225d2ac8f35',
  "active": true,
  "traffic_light_color": 3,   # Color green
  "divisible_by": [3, 5, 11],
  "reason": 'server down'
}
# ~~~
```

3.1. Representation of Variants

The optional keyword (Section 2.1.1 of [RFC9420]) is represented as a struct. For example given the following structs:

```
struct {
  uint16 component_id;
  opaque data;
} Component;

struct {
  uint64 epoch;
  optional<Component> component;
} Foo;
```

An instance of Foo which has an optional Component would be represented by the following snippet of EDN:

```
{
  "epoch": 42,
  "component": {
    "present": true,
    "value": {
      "component_id": 0xBABA,
      "data": h''
    }
  }
}
```

When absent it would be represented as:

```
{
  "epoch": 42,
  "component": {
    "present": false
  }
}
```

In general, the presence of a select construction in TLS PL indicates the conditional presence of additional values in the struct. For example, the struct below:

```
{
  LeafNodeSource leaf_node_source;
  select (LeafNode.leaf_node_source) {
    case key_package:
      Lifetime lifetime;
    case update:
      struct{};
    case commit:
      opaque parent_hash<V>;
    };
  Extension extensions<V>;
} PartialLeafNode;
~~
```

would be represented as follows in EDN:

```
~~~ cbor-diag
{
  "leaf_node_source": 1, # enum LeafNodeSource.key_package
  "lifetime": {
    "not_before": 1777979674, # 05-May-2026T13:14:34Z
    "not_after":  1780312474  # 05-Jun-2026T13:14:34Z
  },
  "extensions": []
}
```

The notation struct {} in TLS PL refers to the absense of a value. It is ignored in the representation of an EDN instance.

4. Examples

4.1. MLS LeafNode

The following detailed example represents the LeafNode struct from Section 7.2 of [RFC9420] and embedded structs. It includes additional types from [I-D.ietf-mls-extensions], [I-D.ietf-mimi-room-policy], and [I-D.mahy-mls-sd-cwt-credential].

Given the following TLS structures:

```
struct {
    HPKEPublicKey encryption_key;
    SignaturePublicKey signature_key;
    Credential credential;
    Capabilities capabilities;
    LeafNodeSource leaf_node_source;
    select (LeafNode.leaf_node_source) {
        case key_package:
```

```
            Lifetime lifetime;
        case update:
            struct{};
        case commit:
            opaque parent_hash<V>;
    };
    Extension extensions<V>;
    /* SignWithLabel(., "LeafNodeTBS", LeafNodeTBS) */
    opaque signature<V>;
} LeafNode;

enum {
    reserved(0),
    key_package(1),
    update(2),
    commit(3),
    (255)
} LeafNodeSource;

struct {
    ProtocolVersion versions<V>;
    CipherSuite cipher_suites<V>;
    ExtensionType extensions<V>;
    ProposalType proposals<V>;
    CredentialType credentials<V>;
} Capabilities;

struct {
    uint64 not_before;
    uint64 not_after;
} Lifetime;


struct {
    ExtensionType extension_type;
    opaque extension_data<V>;
} Extension;

struct {
  CredentialType credential_type;
  select (Credential.credential_type) {
...
    case sd_jwt:
      SdJwt sd_jwt;
    };
} Credential;

struct {
```

```
        Bool compacted;
        select (compacted) {
            case true:
                opaque protected<V>;
                opaque payload<V>;
                opaque signature<V>;
                SdJwtDisclosure disclosures<V>;
                opaque sd_jwt_key_binding<V>;
            case false:
                opaque sd_jwt_kb<V>;
        };
    } SdJwt;

    enum {
      false(0),
      true(1)
    } Bool;

    struct {
        ComponentID component_id;
        opaque data<V>;
    } ComponentData;

    struct {
        ComponentData component_data<V>;
    } AppDataDictionary;
    AppDataDictionary app_data_dictionary;

    opaque HPKEPublicKey<V>;
    opaque SignaturePublicKey<V>;

    /* See the "Messaging Layer Security (MLS)" IANA registries for values */
    uint16 CipherSuite;
    uint16 ExtensionType;
    uint16 ProposalType;
    uint16 CredentialType;
    uint16 ComponentID;
```

below is an example instance represented in EDN:

```
# ~~~ tls LeafNode
{
  "encryption_key": h'41542cd14e930f0d6db59743e2a27974
                      4d774ddc9d53679e78741787be96e832',
  "signature_key":  h'2d13ead305b4fbae97a9677a7fc04d2a
                      1778ef843121f8ffdb46c7389578768d',
  "credential": {
    "credential_type": 0x0006,  # JWT Credential Type
```

```
      "compacted": true,
      "protected": '{
        "alg": "ES256",
        "kid": "https://provider.example/jwk"
      }',
      "payload": '{
        "iss": "https://provider.example",
        "sub": "mimi://provider.example/u/alice",
        "aud": "mimi://hub.example/r/clubhouse",
        "exp": 1780312474,
        "cnf": {"jkt": "kPrK_qmxVWaYVA9wwBF6Iuo3vVzz7TxHCTwXBygrS4k"}
      }',
      "signature": h''
      "disclosures": [],
      "sd_jwt_key_binding": h'cac12ac8b387035dbdbeeed8b085d0f1
                              79f12cb1fa81218432268de12365c86c'
    },
    "capabilities": {
      "versions": [1], # ProtocolVersion mls10
      "cipher_suites": [1, 2, 3, 7],
      "extensions": [
        0x0006,   # app_data_dictionary
        0x0007,   # supported_wire_formats
        0x0008    # required_wire_formats
      ],
      "proposals": [0x0008, 0x0009, 0x000A],
      credentials: [0x0006]
    },
    "leaf_node_source": 1, # enum LeafNodeSource.key_package
    "lifetime": {
      "not_before": 1777979674, # 05-May-2026T13:14:34Z
      "not_after":  1780312474  # 05-Jun-2026T13:14:34Z
    },
    "extensions": [
      {
        "extension_type": 0x0006,  # app_data_dictionary extension
        "extension_data": [
          {
            "component_id": 0x0001, # app_components extension
            "data": [
              0x0022,   # participant_list component
              0x0024,   # mls_operational_policy component
              0x0025,   # roles-list component
              0x0027    # base_room_policy component
            ]
          },
          {
            "component_id": 0x0003, # content_media_types extension
```

```
            "data": ["application/mimi-content"]
          }
        ]
      },
      {
        "extension_type": 0xBABA.  # GREASE extension
        "extension_data": h''
      },
      {
        "extension_type": 0xDADA.  # GREASE extension
        "extension_data": h''
      }
    ],
    "signature": h'88ad994cabf15c4e90d0a4af214b1d75
                   7aecef23efffbd9d8e468866cccebac4'
  }
```

5. Security Considerations

TODO Security

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[I-D.ietf-cbor-edn-literals]
Bormann, C., "CBOR Extended Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-23, 29 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-23>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

7.2. Informative References

[I-D.ietf-mimi-protocol]
Barnes, R., Hodgson, M., Kohbrok, K., Mahy, R., Ralston, T., and R. Robert, "More Instant Messaging Interoperability (MIMI) using HTTPS and MLS", Work in Progress, Internet-Draft, draft-ietf-mimi-protocol-06, 25 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-protocol-06>.

[I-D.ietf-mimi-room-policy]
Mahy, R., "Room Policy for the More Instant Messaging Interoperability (MIMI) Protocol", Work in Progress, Internet-Draft, draft-ietf-mimi-room-policy-03, 18 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-mimi-room-policy-03>.

[I-D.ietf-mls-extensions]
Robert, R., "The Messaging Layer Security (MLS) Extensions", Work in Progress, Internet-Draft, draft-ietf-mls-extensions-09, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-mls-extensions-09>.

[I-D.mahy-mls-sd-cwt-credential]
Mahy, R., "Messaging Layer Security Credentials using Selective Disclosure JSON and CBOR Web Tokens", Work in Progress, Internet-Draft, draft-mahy-mls-sd-cwt-credential-01, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-mahy-mls-sd-cwt-credential-01>.

[RFC7094] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <https://www.rfc-editor.org/rfc/rfc7094>.

[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>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

Acknowledgments

TODO acknowledge.

Author’s Address

Rohan Mahy
Email: rohan.ietf@gmail.com