Hello CBOR-LD

An introduction to the Concise Binary Object Representation for Linked Data
The Problem

VCs and DID Docs are too damn big!*

* For a few important Verifiable Credential (VC) and Decentralized Identifier (DID) use cases - QR Codes, efficient storage, offline interactions.
Use Case: Present Verifiable Credential
Example of the Problem

JSON-LD Document
1,217 bytes

LoFi QR Code Display Limit
400 bytes
Goal

Compress JSON-LD* Documents using a general algorithm.

* This mechanism will not work for JSON-only documents for reasons explained later in the presentation
CBOR-LD Results

*Example data used: https://github.com/digitalbazaar/cborld/blob/main/examples/cit.jsonld
Verifiable Credentials on Low Fidelity Displays

JSON-LD Verifiable Credential
1,217 bytes

CBOR-LD Verifiable Credential
325 bytes
How does it work?
How Compression Works

Reduce data duplication
aka: Don't Repeat Yourself (DRY)

The key to good compression
CBOR-LD (no compression)

No dictionary. Just convert directly to CBOR.

```json
{
  "@context": "https://www.w3.org/ns/activitystreams",
  "type": "Note",
  "summary": "A note",
  "content": "This is an example note."
}
```

JSON-LD Document

140 bytes

Raw CBOR-LD Document

111 bytes

(20% smaller)
CBOR-LD Semantic Compression

Build compression dictionary using @context.

Globally linkable, industry-standard dictionary

No need to include the dictionary in data!

<table>
<thead>
<tr>
<th>CBOR-LD Compression Dictionary for</th>
<th>0x01</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td></td>
</tr>
<tr>
<td>issuanceDate</td>
<td>0x02</td>
</tr>
<tr>
<td>issuer</td>
<td>0x03</td>
</tr>
<tr>
<td>VerifiableCredential</td>
<td>0x04</td>
</tr>
<tr>
<td>... and so on...</td>
<td></td>
</tr>
</tbody>
</table>
CBOR-LD Semantic Compression

Semantic compression better than binary compression.

```
{
    "@context": "https://www.w3.org/ns/activitystreams",
    ...
}
```

53 bytes

01          // {Key:0}, 1 - @context
10          // {Val:0}, 16 - https://www.w3.org/ns/activitystreams

2 bytes

This is the **key innovation** for getting compression ratios that are far **better than** pure binary compression.
Term Compression

Compress known JSON-LD terms.

```json
{
    "type": "VerifiablePresentation",
    ...
}
```

34 bytes

18 // Positive number, next 1 byte
2b // {Key:2}, 43 - type
0e // {Val:2}, 14 - VerifiablePresentation

3 bytes
DateTime Compression

Compress xsd:dateTime values.

```json
{
    "issuanceDate": "2020-07-14T19:23:24Z",
    ...
}
```

39 bytes

18 // Positive number, next 1 byte
1e // {Key:3}, 30 - issuanceDate
1a // Positive number, next 4 bytes
5f0e062c // {Val:3}, 1594754604 - 2020-07-14T19:23:24Z

7 bytes
Binary Compression

Compress Multibase-encoded fields to raw bytes.

```json
{
  "proofValue": "M55Q1ewxSHq5kS...Ui23IFCWA==",
  ...
}
```

107 bytes

18  // Positive number, next 1 byte
26  // {Key:2}, 38 - proofValue
58  // Bytes, length next 1 byte
40  // Bytes, length: 64
   e794357b...14258 // {Val:2} - EdDSA signature

68 bytes
URL Compression

Convert well-known URL patterns to binary.

```json
{
    "issuer": "did:key:z6MkhNZxXHvf4YMbZkEGkA9QA6gN8fP47EdCEJMF5Hh",
    ...
}
```

73 bytes

18  // Positive number, next 1 byte
20  // {Key:2}, 32 - issuer
82  // {Val:2}, Array, 2 items
19  // Positive number, next 2 bytes
0400 // [0], 1024 - did:key
58  // Bytes, length next 1 byte
22  // Bytes, length: 34
    ed012b5f69...1cb9fa628 // [1] - (Ed25519 public key)

41 bytes
CBOR-LD Extensibility

Extensible encoders and decoders (codecs):

- Global URL/Datatype codecs (base64url DIDs?)
- Application-specific @context URLs
- Application-specific URL/Datatype codecs
Benefits of CBOR-LD

- Compression ratios up to 40% better than gzip/zlib.
- Works on any compacted JSON-LD Document.
- LD Proof digital signatures do not need to change.
- Extensible registry-based codec model.
- Application-specific codecs.
- "Practical" set of design trade-offs.
Drawbacks of CBOR-LD

- As of July 2020 - it's new, expect bugs.
- Doesn't work for JSON-only data.
- Only works on compacted JSON-LD.
- Processing overhead on top of JSON-LD.
- Codecs are "more complex than necessary".
- Not as compact as Artisanal CBOR.
# CBOR-LD Results

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON-LD (human readable)</td>
<td>1217</td>
</tr>
<tr>
<td>JSON-LD (jwt)</td>
<td>1009</td>
</tr>
<tr>
<td>JSON-LD (no spaces)</td>
<td>935</td>
</tr>
<tr>
<td>CBOR-LD (raw)</td>
<td>869</td>
</tr>
<tr>
<td>JSON-LD (jwt+gzip)</td>
<td>682</td>
</tr>
<tr>
<td>JSON-LD (gzip)</td>
<td>560</td>
</tr>
<tr>
<td>LoFi QR Code Limit</td>
<td>400</td>
</tr>
<tr>
<td>CBOR-LD (sc+gzip)</td>
<td>329</td>
</tr>
<tr>
<td>CBOR-LD (sc)</td>
<td>325</td>
</tr>
<tr>
<td>Artisanal CBOR</td>
<td>284</td>
</tr>
</tbody>
</table>

*CBOR-LD with Semantic Compression*
The Vision for CBOR-LD

- Eventually, byte-level semantic processing
- Go to CBOR-LD and stay there
- Semantic processing over fixed data structures
- Smaller data means faster calculations
- Push ability to reason (AI) to CPU register level
Questions?
Appendix
Links to Projects

- Implementation
  - https://github.com/digitalbazaar/cborld

- CBOR-LD Specification
  - https://digitalbazaar.github.io/cbor-ld-spec/