

Verifying Digital Signatures on California DL/ID Documents



Technical Overview
December 2025

Verifying Digital Signatures on California DL/ID Documents

This document is intended for technical audiences interested in implementing the technologies listed herein; those interested in using an existing verifier to check the security features on a DL/ID document can do so by visiting <https://verify-credentials.dmv.ca.gov/>.

This document references the following specifications:

- [Verifiable Credential Barcodes 0.7](#)
- [CBOR-LD 1.0](#)
- [Bitstring Status List 1.0](#)
- [Verifiable Credential Data Integrity 1.0](#)
- [Verifiable Credentials Data Model 2.0](#)
- [Decentralized Identifiers 1.0](#)

Overview

This purpose of this document is to provide a technical introduction into how the digital signature security feature on the 2025 California DL/ID Card Design can be checked when scanning the PDF417 barcode.

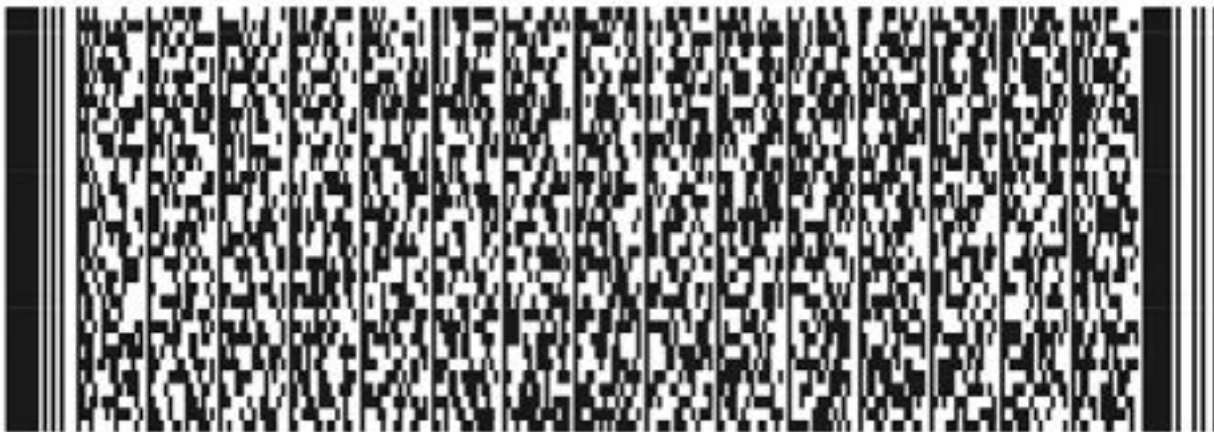
At a high level, the process is as follows:

1. Extract the digital signature payload from the PDF417.
2. Decode the compressed CBOR-LD bytes back to a JSON-LD Verifiable Credential.
3. Process the other data in the PDF417 to construct the data required to verify the digital signature.
4. Perform a Data Integrity cryptographic verification.
5. Check the Bitstring Status List status entry on the credential.

If the cryptographic verification operation succeeds and the status check indicates the DL/ID has not been revoked, the physical document has passed the advanced security check.

Extracting the digital signature payload from the PDF417

The following Driver's License PDF417 will be used as a running example in this document. This PDF417 is for test purposes only.



Reading the data from the PDF417 yields the following:

```
b'@\n\x1e\rANSI
636014100102DL00410287ZC03280220DLDAQI8887059\nDCSFIVENINE\nDDEU\nDACTOMTEST\nDDFU\nDADNONE\nDDGU\
nDCABM1\nDCBL80\nDCDNONE\nDBD08052025\nDBB01011980\nDBA01012030\nDBC1\nDAU070 IN\nDAYBLU\nDAG2415
1ST AVE 051\nDAISACRAMENTO\nDAJCA\nDAK958180000 \nDCF08/05/2025298TL/TLFD/30\nDCGUSA\nDAZBLN\
nDAW200\nDCUIV\nDCK25217I88870590401\nDDAN\nDDB05202024\rZCZCABLU\nZCBBLN\nZCC\nZCD\nZCE2csdghoB2QX
ApgGCAQIYnYIYdhikGK6jGJwYphjEQRUYxhrrmdn0HGLCiGJwYoBioRHX/cGAYtEEUGLa1GJwYbBjMARjWGNwY2FhBepqE+tby
DCJM7XgRUzmFAY41xQu1cBinSYoLAAx/k6284STfQ0g1arQCtDpTWsq96BLDbWzvj0R4JIrk/K5AAY2kEW\r'
```

In California licenses, the signatures are in the **ZCE** field of the **ZC** subfile:

```
2csdghoB2QXApgGCAQIYnYIYdhikGK6jGJwYphjEQRUYxhrrmdn0HGLCiGJwYoBioRHX/cGAYtEEUGLa1GJwYbBjMARjWGNwY2F
hBepqE+tbyDCJM7XgRUzmFAY41xQu1cBinSYoLAAx/k6284STfQ0g1arQCtDpTWsq96BLDbWzvj0R4JIrk/K5AAY2kEW
```

Decoding the Compressed CBOR-LD Bytes Back to a JSON-LD Verifiable Credential

California signatures are base64 encoded. Removing that encoding and expressing as hex:

```
d9cb1d821a01d905c0a601820102189d82187618a418aea3189c18a618c4411518c61ae6767d0718b0a
2189c18a018a84475ff706018b4411418b6a5189c186c18cc0118d618dc18d858417a986a13eb5bc8308933b5e0454c
e6140638d7142ed5c0629d26282c0031fe4eb6f384937d0d20d5aad0ad0e94d6b2a6bde812c36d6cef8f4478248ae4f
cae40018da4116
```

This CBOR-LD can then be converted back to JSON-LD (CBOR-LD spec, Javascript implementation).

California DL/ID CBOR-LD payloads use [registry entry 31000000](#), so the CBOR-LD processor will need that metadata passed to it to decode correctly.

Appendix A of this document contains an example script that uses the above implementation to correctly decode the payload. This results in the following JSON-LD Verifiable Credential:

```
{
  '@context':
  [
    'https://www.w3.org/ns/credentials/v2',
    'https://w3id.org/vc-barcodes/v1'
  ],
  credentialStatus: {
    terseStatusListBaseUrl: 'https://api.uat-credentials.dmv.ca.gov/status/dlid/1/status-lists',
    terseStatusListIndex: 3866524935,
    type: 'TerseBitstringStatusListEntry'
  },
  credentialSubject: {
    protectedComponentIndex: 'u_3Bg',
    type: 'AamvaDriversLicenseScannableInformation'
  },
  issuer: 'did:web:uat-credentials.dmv.ca.gov',
  proof: {
    cryptosuite: 'ecdsa-xi-2023',
    proofPurpose: 'assertionMethod',
    proofValue: 'z43jxJrDSKCMo83ki5FWSjV7CTugPzco4g4xT2X2RegvyjJwpRS59U2nHbvdtLo9R9Xejvy7HqwaYU6Ngn-
HJEAo6F',
    type: 'DataIntegrityProof',
    verificationMethod: 'did:web:uat-credentials.dmv.ca.gov#vm-vcb-1'
  },
  type: [ 'VerifiableCredential', 'OpticalBarcodeCredential' ]
}
```

Processing the Data in the PDF417 to Prepare to Verify the Digital Signature

The digital signature in the Verifiable Credential does not only sign over the credential itself - it also signs over other data from the PDF417.

To prepare for cryptographic verification, apply [this algorithm](#) to the PDF417 contents to construct *opticalDataHash*.

From our example, using the fields corresponding to a *protectedComponentIndex* value of `u_3Bg`:

```
canonicalizedData = 'DACTOMTEST\nDADNONE\nDAG2415 1ST AVE  
051\nDAISACRAMENTO\nDAJCA\nDAK958180000 \nDAQI8887059\nDAU070  
IN\nDBA01012030\nDBB01011980\nDBC1\nDCGUSA\nDCSFIVENINE\n'
```

NOTE: There is a “double hash” that occurs here for security purposes – *canonicalizedData* should be hashed to construct *opticalDataBytes*, which will then need to be hashed again to construct *opticalDataHash*.

[Example DL/ID PDF417 processing from the Verifiable Credential Barcodes specification](#)

Performing Cryptographic Verification

Once you have processed the barcode to construct *opticalDataHash*, a [Verifiable Credential Data Integrity verification](#) with the [ecdsa-xi-2023 cryptosuite](#) can be performed, which is essentially an [ecdsa-rdfc-2019 verification](#) with the addition of *opticalDataHash*, the hash of the signed data in the PDF417 other than the credential itself.

The location of the key material required for verification can be found in the *credentialSubject.proof.verificationMethod* property. In our example:

```
did:web:uat-credentials.dmv.ca.gov#vm-vcb-1
```

This [Decentralized Identifier](#) indicates that the key used for verification is the one found at <https://uat-credentials.dmv.ca.gov/well-known/did.json> and labeled `vm-vcb-1`.

Production Identifiers

The running example in this document uses test cryptographic material and status infrastructure that differ from what is on production DL/ID documents. The correct values to use for production documents are described here.

In production, verifiers **must** enforce checks at this point that the [Decentralized Identifier](#) in the *issuer* and *credentialSubject.proof.verificationMethod* properties are the expected production California Department of Motor Vehicles Decentralized Identifier:

```
did:web:credentials.dmv.ca.gov
```

This DID document can be retrieved from

```
https://credentials.dmv.ca.gov/well-known/did.json
```

Similarly, production Bitstring Status List locations will have the prefix “[api.credentials.dmv.ca.gov/status/dlid](#)”, and verifiers must enforce that the data in the Verifiable Credential conforms with this requirement during verification.

```
api.credentials.dmv.ca.gov/status/dlid
```

Verifiable Credential Barcodes were first added to California DL/ID documents on 09/29/2025. Implementations must reject documents issued after that date that do not contain a Verifiable Credential Barcode. In addition, implementations must differentiate between an invalid or missing digital signature, a revoked digital signature, and an expired credential.

Checking the Bitstring Status List Status Entry on the Credential

The status entry on our example credential is a [TerseBitstringStatusListEntry](#). This is an application of [Bitstring Status List](#) for status tracking where the expression of the status list and index on the credentials themselves are more compact than in a regular [BitstringStatusListEntry](#).

```

credentialStatus: {
  terseStatusListBaseUrl: 'https://api.uat-credentials.dmv.ca.gov/status/dlid/1/status-lists',
  terseStatusListIndex: 3866524935,
  type: 'TerseBitstringStatusListEntry'
}

```

To [get the status list URL](#), we do the following:

$$listIndex = floor \left(\frac{terseStatusListIndex}{statusListSize} \right) = floor \left(\frac{3866524935}{67108864} \right) = 57$$

$$statusIndex = 3866524935 \% 67108864 = 41319687$$

For status purpose “revocation”, the location of the status list for this credential is then:

<https://api.uat-credentials.dmv.ca.gov/status/dlid/1/status-lists/revocation/57>

Fetch this list, process it per the [Bitstring Status](#) List specification, and check bit 41319687. If this bit is set to 1, the credential has been revoked. Appendix B of this document contains an example PDF417 containing a digital signature that has been revoked.

Appendix A: Example CBOR-LD Decode Interface

Implementation: <https://github.com/digitalbazaar/cborld>

The following is a simple Javascript example of how to use the `decode()` interface in the above implementation of CBOR-LD.

```

-----
// credential data
const hex =
'd9cb1d821a01d905c0a601820102189d82187618a418aea3189c18a618c4411518c61ae6767d0718b0a
2189c18a018a84475ff706018b4411418b6a5189c186c18cc0118d618dc18d858417a986a13eb5bc
8308933b5e0454ce6140638d7142ed5c0629d26282c0031fe4eb6f384937d0d20d5aad0ad0e94d6b2a6bde
812c36d6cef8f4478248ae4fcae40018da4116'
const cborldBytes = Uint8Array.from(Buffer.from(hex, 'hex'));

// prepare CBOR-LD registry entry metadata for decoding
const contextTable = new Map([
  ["https://www.w3.org/ns/credentials/v2", 1],
  ["https://w3id.org/vc-barcodes/v1", 2]
]);

const cryptosuiteTable = new Map([
  ["ecdsa-xi-2023", 1]
]);

const urlTable = new Map([
  ["did:web:credentials.dmv.ca.gov", 1],
  ["https://api.credentials.dmv.ca.gov/status/dlid/1/status-lists", 2],
  ["https://api.credentials.dmv.ca.gov/status/dlid/2/status-lists", 3],
  ["https://api.credentials.dmv.ca.gov/status/dlid/3/status-lists", 4],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-1", 5],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-2", 6],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-3", 7],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-4", 8],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-5", 9],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-6", 10],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-7", 11],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-8", 12],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-9", 13],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-10", 14],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-11", 15],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-12", 16],
  ["did:web:credentials.dmv.ca.gov#vm-vcb-13", 17],

```

```

["did:web:credentials.dmv.ca.gov#vm-vcb-14", 18],
["did:web:credentials.dmv.ca.gov#vm-vcb-15", 19],
["did:web:uat-credentials.dmv.ca.gov", 20],
["https://api.uat-credentials.dmv.ca.gov/status/dlid/1/status-lists", 21],
["did:web:uat-credentials.dmv.ca.gov#vm-vcb-1", 22],
["did:web:uat-credentials.dmv.ca.gov#vm-vcb-2", 23],
["did:web:uat-credentials.dmv.ca.gov#vm-vcb-3", 24],
["did:web:uat-credentials.dmv.ca.gov#vm-vcb-4", 25],
["did:web:uat-credentials.dmv.ca.gov#vm-vcb-5", 26],
["https://api.uat-credentials.dmv.ca.gov/status/dlid/2/status-lists", 27],
["https://api.uat-credentials.dmv.ca.gov/status/dlid/3/status-lists", 28],
]);

const typeTable = new Map();
typeTable.set('https://w3id.org/security#cryptosuiteString', cryptosuiteTable);
typeTable.set('context', contextTable);
typeTable.set('url', urlTable);

// prepare JSON-LD contexts for decoding
const CREDENTIALSV2 = {...} // JSON-LD found at https://www.w3.org/ns/credentials/v2
const VCBV1 = {...} // JSON-LD found at https://w3id.org/vc-barcodes/v1

const documentLoader = url => {
  if(url === 'https://www.w3.org/ns/credentials/v2') {
    return {
      contextUrl: null,
      document: CREDENTIALSV2,
      documentUrl: url
    };
  }
  if(url === 'https://w3id.org/vc-barcodes/v1') {
    return {
      contextUrl: null,
      document: VCBV1,
      documentUrl: url
    };
  }
  throw new Error(`Refused to load URL "${url}"`);
};

// use decode interface
const jsonldDocument = await decode({
  cborldBytes,
  documentLoader,
  typeTableLoader: () => typeTable
});

console.log(jsonldDocument);
-----

```

Appendix B: Example PDF417 with revoked digital signature

