INTERNATIONAL STANDARD

ISO/IEC 20248

Second edition
2022-06

Information technology — Automatic identification and data capture techniques — Digital signature data structure schema

Technologies de l'information — Techniques d'identification automatique et de capture de données — Schéma de structure de données de signature numérique

Reference number
ISO/IEC 20248:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see

This document was prepared by joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 20248:2018), which has been technically revised.

The main changes are as follows:

— The relationship between the Domain Authority (data owner) and the Domain Authority ID (DAID) is clarified to be one-to-many. The DAID has been extended to cater for the GS1 Company Prefix longer than 10 digits (see 7.5.3), and a method to use the primary data carrier DAID, if present (see 7.5.2).

— The data types and specifications have been updated for easier implementation and completeness, especially to support the practice of using the data type specifications to achieve optimized schema-based data encoding. A codebook method forms part of this update.

— The `date` field type has been found to be limiting. A new human readable `isodate` has been specified to replace `date` (see 8.9.3.7).

— The format of binary data is explicitly defined to be `HexString` or `Base64String` ensuring interoperability and ease of use.

— The `bstring` DDDdata has been limited to `HexString` since Base64 decoding can be done in more than one way which may cause a valid DigSig to be rejected.

— The `digsigenv` type has been changed from `bstring` to `string` with a range of `Base64String`, which is technically the same, but explicit and clear.

— The `cidsniptext` pragma (field directive) has been removed since it is not practical, not used, and redundant. It is also difficult and convoluted to use and implement.

— ISO/IEC 9899, *Information technology — Programming languages — C* has been removed as a normative reference. Common current coding language methods replaced the C methods.

— Example cryptography methods are provided in B.4.

— Example interfaces to potential code blocks are provided in D.3.3 and E.3.3.

— Revocation has been harmonized with conventional best practices. The CID requirement to be 0 and 1 has been removed (see Annex J).

— An example implementation architecture description has been added as Annex N.

— The structured document function (see 8.11) has been enhanced to support multiple languages. An example structured document is discussed as Annex O.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document specifies a data structure framework and data specification method for domain-authority-specified, schema-based item identification data. A domain authority is typically a brand owner, a data authority, or a data owner.

ISO/IEC 21778 (JSON) is used as the data message format for both the schema and the data, ensuring interoperability with modern Internet systems and services. The data message encapsulates both data syntax and semantics, providing meaning to the data message.

The data source, data schema and data are both offline and online verifiable using ISO/IEC 9594-8 (public key infrastructure (PKI) digital signatures and certificates), with its implementation environment. The data message format allows for the verification of the data message anywhere within the data-stack.

Data capacity and/or data transfer capacity of automated identification data carriers (barcode labels and RFID tags) are limited. This restricts the normal use of a digital signature, as specified in ISO/IEC 9594-8, within automated identification services. This limitation is overcome by the methods specified in this document, which recognizes the three classes of item; data carrier data (any combination of barcodes and RFID tags), generic data which applies to a group of items, and item specific data which may be static for that item, or volatile. Only item specific data are carried by the tag. Generic data are carried by the digital certificate associated with the tag. This method allows additional (comprehensive) data about a group of items to be readable and verifiable.

Adding additional data, especially authenticity data, to tags are often challenging for existing systems resulting in high costs and system/services unavailability. This document provides a method whereby data may be added with limited impact to incumbent systems, facilitating an interoperable add-on rather than a system redesign.

This document specifies an effective and interoperable method to specify, read, decode, and verify data stored in automated identification, independent from real-time remote control. Meta parameters included in a digital certificate are used to achieve:

— offline integrity verification of the data source and data originality,

— a verifiable data structure description to enable interoperability of deployment, domain authority and automated identification data carriers,

— a verifiable data encoding method to achieve compact data to be stored in data constrained automated identification data carriers (the JSON data format is used for both input and output of the encoder and decoder),

— a verifiable automated identification data carrier read method description, allowing for the data of a read event to be distributed over more than one carrier of the same and of different technologies, and

— a verifiable method to support key management of cryptographically-enabled automatic identification data carriers.

A successful verification of the DigSig signifies:

— the data was not tampered with;

— the source of the data is as indicated on the DigSig certificate used to verify the DigSig with;

— if a secured unique identifier of the data carrier is included in the signature of the DigSig stored on data carrier, then the DigSig stored on the data carrier can be considered unique and original.

The choice of cryptography method should be considered carefully. It is advised that only internationally recognized or standardized methods, e.g. FIPS PUB 186-4 and IEEE P1363, be used.

This document should be used in conjunction with standard risk assessments of the use-case and environment.

NOTE Many applications rely on a secure non-transferable unique data carrier identifier to tag an item uniquely. ISO/IEC 29167 gives more information on such functionality for RFID tags. This specification provides a mechanism to ensure the integrity and authenticity of the data carrier data and an irrefutable link of the data carrier data with the unique data carrier identifier. As such, alterations or insertion of false data into data carriers are detectable. It also provides a means to detect tampered data carrier data stored and communicated within systems. It does not provide any means to defend against replay attacks. As a counter the data carrier reader can use this specification to sign the read data, effectively providing integrity and authenticity to the read-transaction. A third party can then verify that the read-transaction happened at a given place and time, as well as verify the data read from the carrier. Likewise, the signed data carrier data can contain data describing unique features and security marks of the item establishing a verifiable link between the data carrier data and the physical item.

Information technology — Automatic identification and data capture techniques — Digital signature data structure schema

1 Scope

This document is an ISO/IEC 9594-8 [public key infrastructure (PKI) digital signatures and certificates] application specification for automated identification services. It specifies a method whereby data stored within a barcode and/or RFID tag are structured, encoded and digitally signed. ISO/IEC 9594-8 is used to provide a standard method for key and data description management and distribution. The data capacity and/or data transfer capacity of automated identification data carriers are restricted. This restricts the normal use of a digital signature as specified in ISO/IEC 9594-8 within automated identification services.

The purpose of this document is to provide an open and interoperable method, between automated identification services and data carriers, to read data, verify data originality and data integrity in an offline use case.

This document specifies

— the meta data structure, the DigSig, which contains the digital signature and encoded structured data,

— the public key certificate parameter and extension use, the DigSig certificate, which contains the certified associated public key, the structured data description, the read methods, and private containers,

— the method to specify, read, describe, sign, verify, encode, and decode the structured data, the DigSig Data Description,

— the DigSig EncoderGenerator which generates the relevant asymmetric key pairs, keeps the private key secret, and generates the DigSigs, and

— the DigSig DecoderVerifier which, by using to the DigSig certificate, reads the DigSig from the set of data carriers, verifies the DigSig and extracts the structured data from the DigSig.

This document does not specify

— cryptographic methods, or

— key management methods.

2 Normative references

The following documents are referred to in the text in such a way that some or all their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8601 (all parts), *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 8824-1[1], *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation — Part 1*

[1] ITU-T X.680 is equivalent to ISO/IEC 8824-1.

ISO/IEC 9594-1[2], *Information technology — Open Systems Interconnection — The Directory — Part 1: Overview of concepts, models and services*

ISO/IEC 9594-8[3], *Information technology — Open Systems Interconnection —The Directory — Part 8: Public-key and attribute certificate frameworks*

ISO/IEC/IEEE 9945, *Information technology — Portable Operating System Interface (POSIX®) Base Specifications, Issue 7*

ISO/IEC 19762, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

IETF RFC 5646[4], *Tags for Identifying Languages*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
authenticity
quality or condition of being authentic, trustworthy, or genuine

3.2
base64url
Base64 encoding with the URL and Filename Safe Alphabet

Note 1 to entry: See IETF RFC 4648.

3.3
CIDSnip
continuous binary sequence starting with the DAID and CID encoded field values

Note 1 to entry: See 8.3.2.

3.4
DataSnip
continuous binary or text sequence of encoded field values

Note 1 to entry: See 8.3.2.

3.5
digital certificate
certificate
data construct that contains the public key, integrity parameters and use parameters of the DigSig

Note 1 to entry: The data construct shall be as specified in ISO/IEC 9594-8.

[2] ITU X.500 is equivalent to ISO/IEC 9594-1, and is the commonly used reference for standard and terminology.

[3] ITU X.509 is equivalent to ISO/IEC 9594-8, and is the commonly used reference for standard and terminology.

[4] IEFT RFC 5646 is the reference specification of the IETF BCP 47.

3.6
digital signature
signature
result of an asymmetric encryption method on a data construct

Note 1 to entry: The asymmetric encryption method and data construct shall be as specified in ISO/IEC 9594-8.

Note 2 to entry: In typical legal terminology, this term is the equivalent of an advanced electronic signature.

3.7
DigSig
data construct assembled according to this document which contains verifiable information obtained from one or more AIDC

3.8
DigSig envelope
envelope
data construct assembled according to this document by the EncoderGenerator

3.9
Domain Authority
entity, operating as a trusted third party, responsible for the digital signature integrity of a jurisdiction

3.10
integrity
reliability of data that are as they were created according to the required verification parameters

3.11
jurisdiction
independent domain of control in terms of the business or legal (or both) scope of the parties concerned

Note 1 to entry: Examples are independent countries, separate ministries or departments of a government, or independent companies each with their own legal or business (or both) framework.

3.12
nibble
four-bit aggregation

3.13
pragma
field directive

Note 1 to entry: See Clause 9.

3.14
Private key
key that is kept in secret and is used to generate a digital signature by encrypting data that will be verified by its associated public key

3.15
Public key
key that is publicly available and is used to verify data that were encrypted by its associated private key

3.16
Snip
continuous binary or text sequence

Note 1 to entry: See 8.3.2.

3.17
UTF-8
8 bit variable-width encoding as specified by ISO/IEC 10646 of the Unicode Graphic code points

Note 1 to entry: See ISO/IEC 10646.

3.18
WORD
media physical memory grouping of bits

4 Field and data definitions, abbreviated terms, symbols, and binary data

4.1 Field and data definitions

Field and data objects are defined in Clause 8 and 8.10.3.

4.2 Abbreviated terms

AFI	Application Family Identifier
AIDC	Automatic Identification data capture
BRE	Basic Regular Expression
CA	Certification Authority
CID	DigSig certificate ID
DA	Domain Authority
DAID	Domain Authority identifier
DDD	DigSig Data Description
DI	Data Identifier (see ISO/IEC 15434)
DigSig IA	DigSig Issuing Authority
ERE	Extended Regular Expressions
ID	Identification number
IoT	Internet of Things
JSON	Data description construct (see ISO/IEC 21778)
MSB	Most significant bit
OID	Object identifier as specified in ISO/IEC 8824-1
PKI	Public key infrastructure
RFID	Radio-frequency identification
UID	Unique ID
URI	Uniform Resource Identifier
UTC	Coordinated Universal Time

X.509	ISO/IEC 9594-8

4.3 Symbols

||	concatenate or join
...	repeat the previous, as required, or fill in, as required
{...}	parameters that form one structure/grouping, a JSON object
[x]	x is optional, or a JSON array
<x>	x is compulsory
x |= y	y is a description of x
x ⇐ y	x takes the value of y
# x	x is a comment until the end of the line
F(x,y)	the function F that takes as input two parameters, x and y, to produce an output
:XXXX:..XX	depicts a hexadecimal string with "0" to "9" and "A" to "F" with 4-character groupings
0xX...	depicts a hexadecimal value with X with "0" to "9" and "A" to "F"
XXX_2	"XXX" is binary with X a "0" or a "1"

4.4 Binary data

Binary data shall be represented by a JSON string with one of the following two formats:

- `HexString`: The character 0 to 9 and A to F shall be used to represent binary values 0000_2 to 1111_2. A `HexString` is a continuum of 4-bit nibbles presented as uppercase hex grouped with 4 characters preceded by a colon (":"). Zero-bit padding shall be used to reach a 4-bit boundary.

  EXAMPLE 1 `":0123:4567:89AB:CDEF:0123:4567:89AB:C"` and `":92"`

- `Base64String`: IETF RFC 4648 base64url with padding shall be used except when `Base64String` is used as an input value for bstring where multiples of 6 bits are represented each with a base64url character.

  EXAMPLE 2 `"VGhlIHF1aWNrIGJzyQMjezuZcA=="`

Binary data shall be represented in network byte order (big-endian on bytes).

Applications shall auto distinguish between `HexString` and `Base64String` by using the leading colon (":") character in the JSON string.

5 Conformance

5.1 Specification version

The specification version is used by data structures defined in this document. Other systems use the specification version to identify the data structures of this document and to determine the version of the specification.

The specification version shall be set as follows:

specificationversionvalue ⇐ "ISO/IEC 20248:yyyy" with "yyyy" the year of the publication supported by the implementation.

5.2 Claiming conformance

To claim conformance, a service shall comply with the requirements of this document.

5.3 Test authority

The tests shall be performed by a software test authority using a norm application or by code inspection. The norm application used in this test shall be independent from the person who requests the test.

5.4 Test specification

The test specification specifies the conformance test methods for this document.

The test methods in Annex A shall be used.

The test specification is independent of

- cryptography conformance and performance; and
- AIDC data carrier conformance and performance.

The following components shall be tested:

- DigSig certificate format.
- DigSig data.
- DigSig DecoderVerifier.
- DigSig EncoderGenerator.

6 DigSig use architecture

6.1 General

This document specifies a DigSig EncoderGenerator and a DigSig DecoderVerifier system component. The DigSig EncoderGenerator is typically an application dedicated implementation and the DigSig DecoderVerifier an application independent implementation.

A DigSig is a structured set of AIDC data. A DigSig may be stored over more than one AIDC device of different types. A DigSig is cryptographically verifiable. The DigSig data structure definition, read methods and cryptographic functions (DigSigs Data Description - DDD) are specified by a Domain Authority (DA) and published in a DigSig certificate (a version 3 X.509 digital certificate). The DigSig certificate is cryptographically verifiable as certified by an X.509 Certification Authority. Each DigSig certificate has a unique identifier (see 7.6) called the certificate Identifier (CID). The {DAID, CID} is unique and contained in every DigSig as the first data of the DigSig. A reader of a DigSig uses the {DAID, CID} of the specific DigSig to reference the relevant DigSig certificate, from which the reader acquires the read methods, data structure specification and cryptographic functions to read the full DigSig, decode the DigSig and verify the DigSig. See Annex B for example DigSigs and Annex I for the DigSig data structures of an example DigSig.

The DigSig EncoderGenerator is used to generate a DigSig, on request from a data carrier programming application. The DigSig EncoderGenerator does not include the method to create or program a data carrier. See Annex D for a typical DigSig EncoderGenerator use architecture.

The DigSig DecoderVerifier is used by a local application to instruct a data carrier reader/interrogator how to read the DigSig from a set of data carriers and other sources to decode and verify the data. See Annex E typical DigSig DecoderVerifier use architecture.

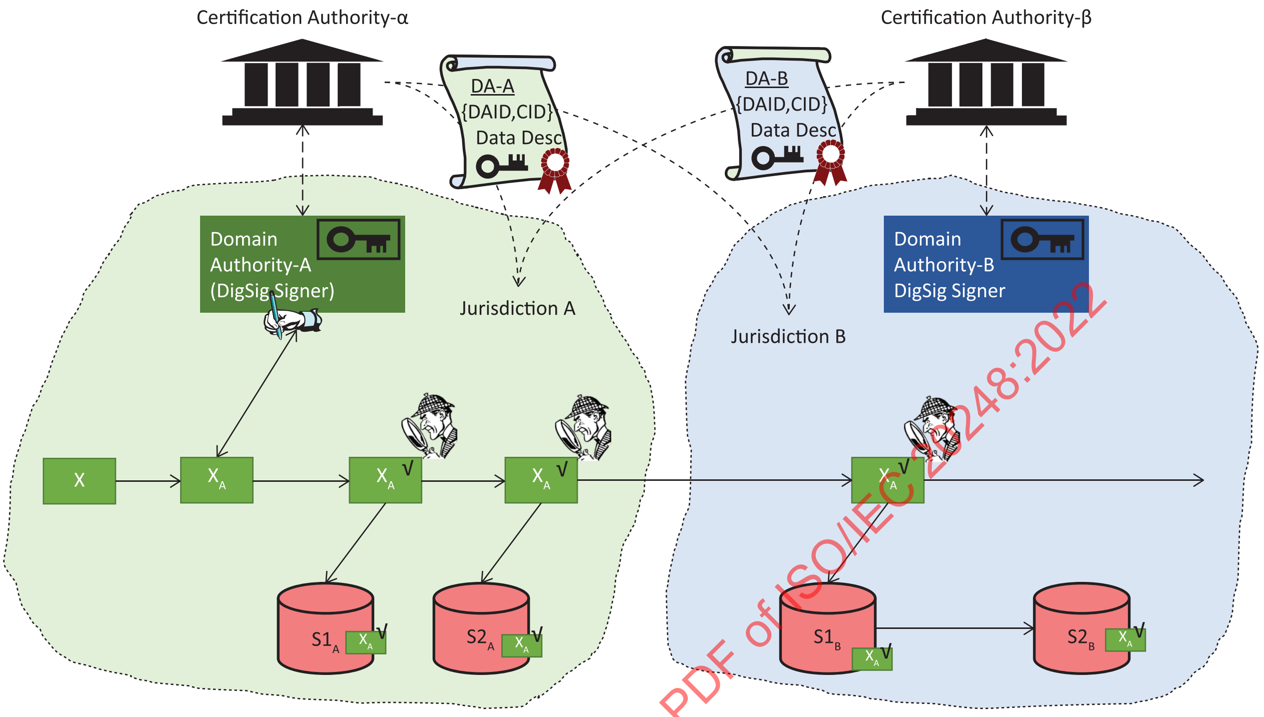


Figure 1 — DigSig use architecture

The general method and properties of this document are illustrated in Figure 1.

a) Domain Authority A (DA-A) provides DigSig issuing services for Jurisdiction A. Similarly, Domain Authority B (DA-B) provides DigSig issuing services for Jurisdiction B. “Issuing” entails the validation of the data for a DigSig, the validation of the DigSig requestor's credentials, and the generation of the DigSig. The DigSig requestor may be a human and/or an application.

b) The DigSig certificates issued by DA-A, each containing a DigSig Data Description (DDD) as applicable to Jurisdiction A applications/services, are certified by the Certification Authority α (CA-α) for a specific signing and certificate validity period in accordance with a Certification Practice Statement as specified in X.509. Similarly, Certification Authority β certifies DigSig certificates issued by DA-B. Certification Authorities α and β may be the same entity.

c) The DigSig certificates are published in a manner to allow systems $S1_A$, $S2_A$, $S1_B$, $S2_B$... to acquire them in advance or on demand. The DigSig certificates are used by the systems to read, decode, and verify DigSigs generated and stored on data carriers by the Domain Authorities, e.g. Data X is used by DA-A to generate a DigSig N X_A as specified by a DigSig certificate N. The systems of both jurisdictions use the DigSig certificates to read, decode, verify, and use the data without the need to connect to any other system.

Annex C provides more information on DigSig use in IoT. AIDC data fulfil an important role in IoT by providing physical objects with a digital identity and optional attributes.

Annex G provides more information on digital signature use in general.

6.2 DigSig identification and ownership

The Domain Authority (DA) shall be the owner of the DigSig certificate which specifies the DigSig data structure schema (the DDD). The DA shall be the issuer of the DigSigs, directly or by proxy, specified by the associated DigSig certificate.

The Domain Authority shall be identified by its URI called the DA URI (see 7.3). The DA URI shall be verified by the X.509 Certificate Authority when issuing the DigSig certificate.

The DigSig ownership shall be identified by the tuple {Domain Authority URI (`dauri`), Domain Authority identifier (DAID, see 7.5), DigSig certificate identifier (CID, see 7.6)} as certified by the DigSig certificate to be valid at the time of the issuance of the certificate. This validity should be valid when a DigSig is issued.

A Domain Authority may be related to more than one DAID. A DAID shall only be related to one Domain Authority at a given time.

NOTE A DAID is typically assigned to a company or an operating entity of a company. These entities can be traded, terminated, suspended, and resurrected as part of normal business operations.

6.3 DigSig certificate process

Figure 2 describes the process that shall be followed to issue a DigSig certificate in accordance with X.509.

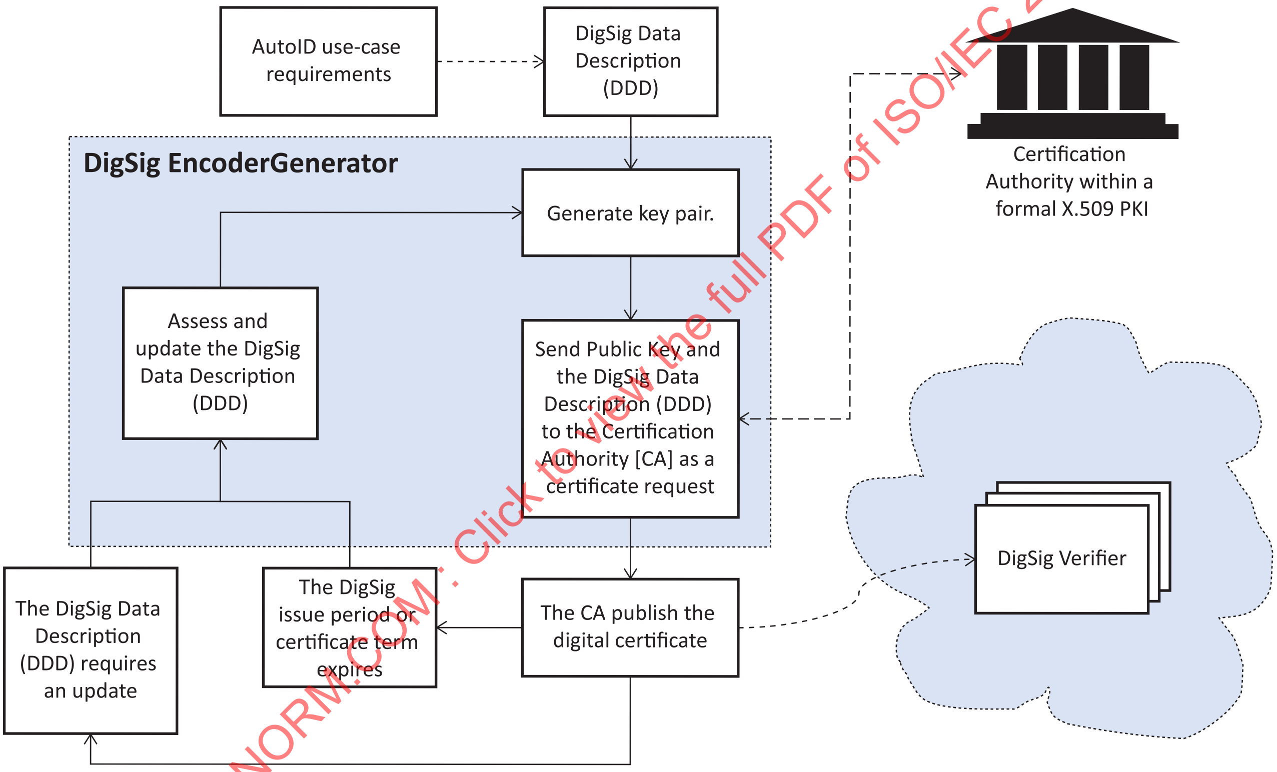

AutoID use-case requirements
DigSig Data Description (DDD)
DigSig EncoderGenerator
Generate key pair.
Certification Authority within a formal X.509 PKI
Assess and update the DigSig Data Description (DDD)
Send Public Key and the DigSig Data Description (DDD) to the Certification Authority [CA] as a certificate request
DigSig Verifier
The DigSig Data Description (DDD) requires an update
The DigSig issue period or certificate term expires
The CA publish the digital certificate

Figure 2 — DigSig certificate process

Figure 3 illustrates how data structure changes can be achieved seamlessly by allowing DigSig certificates — and therefore the data structure validity — to overlap.

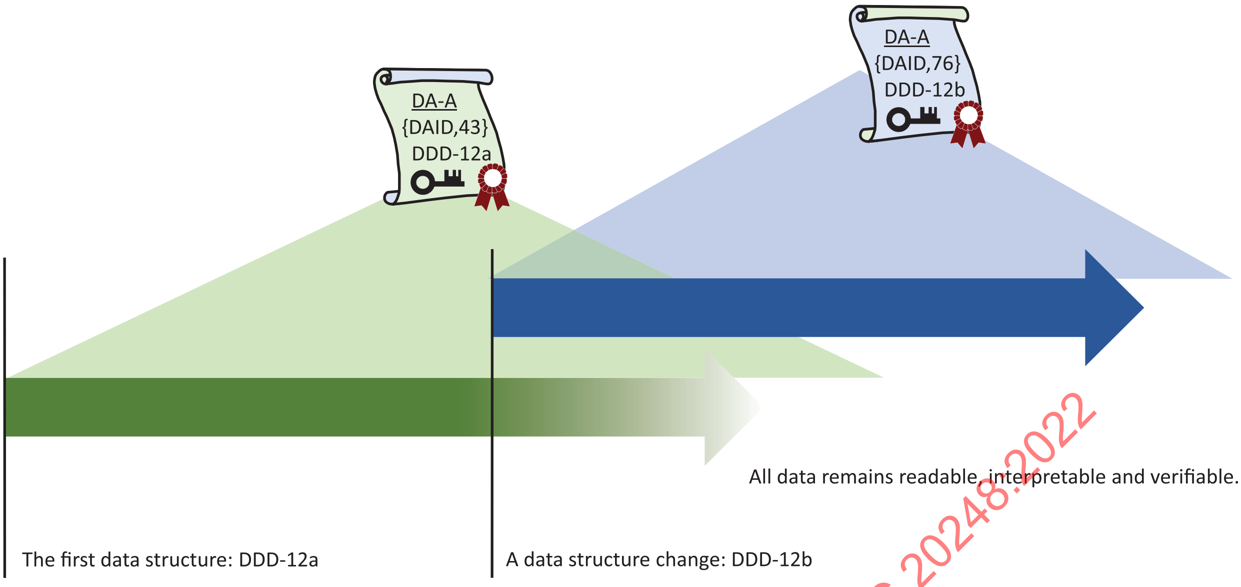

DA-A
{DAID,43}
DDD-12a
DA-A
{DAID,76}
DDD-12b
All data remains readable, interpretable and verifiable.
The first data structure: DDD-12a
A data structure change: DDD-12b

Figure 3 — DigSig data structure rollover

6.4 DigSig generation process

The DigSig is generated by the DigSig EncoderGenerator.

a) The applicable CID referenced DigSig certificate shall be used to generate SigData from DDDdata.

b) SigData shall be digitally signed using the DigSig certificate specified digital signature algorithm resulting in the values Signature and Timestamp.

c) The signature and timestamp DDDdata fields shall be assigned the values Signature and Timestamp.

d) The required DigSig shall be generated from DDDdata.

6.5 DigSig verification process

The DigSig verification shall be performed by a DigSig DecoderVerifier application. The application reads the data carriers as required by its primary function in its use case. It has also the ability to obtain the referenced DigSig certificate.

The steps to verify a DigSig shall be as follows:

a) Read the DigSig envelope.

b) Extract the {DAID, CID} from the DigSig envelope.

c) Check if the {DAID, CID} referenced DigSig certificate is contained in the local certificate store. If not, obtain it from a trusted repository.

d) Verify the {DAID, CID} DigSig certificate, and then use it in the following steps.

e) Read the remainder of the DigSig using the readmethod pragma.

f) Decode the DigSig and prepare DDDdata and SigData.

g) Perform the verification on SigData.

6.6 Error codes

The DigSig error codes shall be in accordance with Annex F.

7 DigSig certificate

7.1 General

The DigSig certificate consists of a X.509 version 3 certificate with the DDD included in the X.509 version 3 extension.

```
DigSig certificate |= <X.509V3 certificate> ‖ <DDD in a X.509V3 extension>
```

The DigSig certificate establishes a verifiable link between

- the Domain Authority (DA) represented by its URI (DA URI with the field `dauri`),
- the Domain Authority Identifier (DAID with the field `daid`),
- the DigSig Data Description (DDD with the fields `digsiginfo` and `datafields`), and
- the authority verification and revocation services (and array of URIs for the DigSig information fields `verificationuri` and `revocationuri`).

The relationship between a DA and a DAID shall be valid at the time of issuance of the DigSig certificate. When that link is broken the DigSig certificate should be revoked.

NOTE 1 A non-hierarchical certification path can be valid when the same DigSig is used by more than one DA.

NOTE 2 It is possible that the desired CA do not support the specific requirements of this document. In this case, an intermediate CA, a DigSig Issuing Authority (DigSig IA), can be included in the PKI, with a singular function of validating the DigSig extension. The desired CA can incorporate the DigSig IA or be the parent of the DigSig IA. X.509 recommends the inclusion of the DigSig IA function into the respective Certification Practice Statement.

7.2 ISO/IEC 20248 Object Identifier

The ISO/IEC 20248 OID shall take the value: ISO.standard."the number of this standard"; this results in the OID: "1.0.20248".

The ISO/IEC 20248 arch, "1.0.20248.1" shall be the container for the ISO/IEC 20248 specific digital signature methods not defined by other standard OIDs (see B.4).

7.3 DigSig certificate parameter use

The X.509 version 3 parameters (in a human readable format with notes) shall be used as follows:

```
Certificate:
  Data:
    Version: 3 (0x2)                        # Shall be version 3
    Serial Number: …                        # set in accordance with X.509
    Signature Algorithm: …                  # set in accordance with X.509
    Issuer: …                               # set in accordance with X.509
    Validity                                # set in accordance with X.509
      Not Before: …
      Not After : …
    Subject: …                              # {DA, DAID, CID} using distinguished names as
                                            #  specified in ISO/IEC 9594-1 and this clause
    Subject public key Info:                # set in accordance with X.509
      Public Key Algorithm: …               # set in accordance with X.509 (see B.4)
      Public-Key: …                         # set in accordance with X.509 (see B.4)
    X509v3 extensions:                      # set in accordance with X.509
      X509v3 Key Usage: critical
```

```
        Digital Signature, Key Encipherment
      X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Authority Key Identifier: …     # set in accordance with X.509
    X509v3 CRL Distribution Points: …      # set in accordance with X.509
    1.0.20248: <DDD>                       # set in accordance with this document
  Signature Algorithm: …                   # set in accordance with X.509
```

The common name of the subject shall have the following format:

```
CN = <DA URI>‖/"daid"/‖<DAID>‖"/cid/"‖<CID>
```

EXAMPLE 1 `CN = https://DomainAuthority.com/daid/QC%20TEST/cid/12345`

EXAMPLE 2 See Annex H.

NOTE 1 A DigSig not aware digital certificate authority (CA) can reject the issuing of the certificate if it does not know the DigSig public key algorithm.

NOTE 2 A DigSig aware digital certificate validator processes the extension and accepts or rejects the DigSig certificate depending on the content of the DigSig certificate, the extension, and the conditions under which processing occurs (e.g. the current values of the path-processing variables).

7.4 DigSig cryptography

7.4.1 General

This document specifies the use of two types of cryptography:

- For digital signatures.
- For private containers.

This document does not specify the specific cryptographic algorithms, key, or hash lengths. Only recognized cryptographic methods shall be used.

NOTE The choice of cryptography is linked to the use case risk profile, the size of memory available on the data carriers to be used, the time available to read/interrogate the data carrier reliably, the scope of the application services (i.e. open vs close loop) and the desired validity period of the DigSig.

7.4.2 Digital signatures

The digital signature cryptography methods (i.e. asymmetric encryption, key length, and hash algorithm) shall be specified within the DigSig certificate in compliance with X.509. B.4 provides example digital signature cryptography methods which may be applicable in the resource constraint AIDC environment.

7.4.3 Private containers

The private container cryptography shall be specified by the `privatecontainer` pragma (see 9.5).

7.5 DigSig Domain Authority identifier (DAID)

7.5.1 Binary encoding

The DigSig Domain Authority identifier (DAID) shall be uniquely constructed using the ISO/IEC 15459-2 Issuing Agency Code (IAC) and the Company Identifying Number (CIN) in the following manner:

```
DAID |= <IAC>‖" "‖<CIN>
```

An ISO/IEC 20248 Domain Authority shall be registered with a CIN.

a) The DAID is constructed as specified in Table 1.

Table 1 — Domain Authority identifier binary encoding

IAC indicator	CIN indicator	IAC code	CIN code
1 bit	2 bits	5 or 13 bits	24 or 32 bits

b) The rules in Table 2 shall be used to encode the IAC.

Table 2 — ISO/IEC 15459-2 IAC encoding

IAC indicator	IAC range	IAC code	Encoding bits
0_2	0-9	0-9	5
0_2	A-J	10-19	5
1_2	LA-UZ	20-279	13
1_2	KAA-KZZ	280-955	13
1_2	VAA-ZZZ	956-4335	13

c) The rules in Table 3 shall be used to encode the CIN. The CIN, both decimal and upper-case alphanumeric, as specified by ISO/IEC 15459-3, is treated as a number with leading zeros or spaces. Examples are provided in Table 4.

Table 3 — ISO/IEC 15459-2 CIN encoding

CIN indicator	CIN type	CIN length	Encoding bits
00_2	Numeric	1-7 digits	24 bits
01_2	Numeric	8-9 digits	32 bits
10_2	Alphanumeric	1-4 char.	24 bits
11_2	Alphanumeric	5-6 char.	32 bits

1) The numeric CIN shall be treated as a Base10 (decimal) number.

2) The upper-case alphanumeric CIN shall be treated as a Base36 number, also known as a hexatridecimal number, encoded as follows:

$$\text{Decimal CIN} \Leftarrow A_n{*}36^n \ldots + A_2{*}36^2 + A_1{*}36 + A_0 \Leftarrow A_n \ldots A_2A_1A_0$$

i) "0" to "9" takes the character values 0 to 9.

ii) "A" to "Z" takes the character values 10 to 35.

iii) Leading zeros and spaces take the character value 0.

NOTE CINs are treated as a number, as such it is padded with leading zeros which is equivalent to leading spaces.

EXAMPLE "123" is equal to "0000123" for 24-bit numerical encoding and "000000123" for 32-bit numerical encoding, and "BC1" is equal to "_BC1" and "0BC1" for 24-bit alphanumerical encoding and "___BC1" and "000BC1" for 32-bit alphanumerical encoding.

Table 4 — DAID encoding examples

15459-2 Example IAC CIN	IAC indicator	CIN indicator	IAC	CIN	DAID calculated from IAC CIN Example
C ZZ	0_2	10_2	01100_2	:00:05:0F	:4C:00:05:0F
C ZZZZ	0_2	10_2	01100_2	:19:A0:FF	:4C:19:A0:FF
B 999	0_2	00_2	01011_2	:00:03:E7	:0B:00:03:E7
B 999999999	0_2	01_2	01011_2	:3B:9A:C9:FF	:2B:3B:9A:C9:FF
4 123456	0_2	00_2	00100_2	:01:E2:40	:04:01:E2:40
QC ZZZZ	1_2	10_2	0000010011000_2	:19:A0:FF	:C0:98:19:A0:FF
QC ZZZZZZ	1_2	11_2	0000010011000_2	:81:BF:0F:FF	:E0:98:81:BF:0F:FF
UN 999999999	1_2	01_2	0000100001011_2	:3B:9A:C9:FF	:A1:0B:3B:9A:C9:FF

7.5.2 Referenced DAID

Where the DigSig is a data element of an AIDC data carrier message, the DAID 0xFF may be used to reference the DAID of the data carrier message.

The CIN of the primary data carrier data shall comply with 7.5.1.

NOTE Annex K a) and Figure M.2 illustrates the inclusion of a DigSig in a data carrier message (primary data carrier data). Often the primary data carrier data, as illustrated, includes a DAID (IAC and CIN). It is common for the issuer of the primary data carrier data and the DigSig signer to be the same entity (Domain Authority) as indicated by the DAID (IAC and CIN). As a result, the overall encoded data message is shortened, which is beneficial for memory and/or communication constraint data carriers.

7.5.3 GS1 Company Prefix (GCP)

GS1 Company Prefixes (GCPs) of length up to 10 digits shall be encoded as specified by 7.5.1 where the first digit is the IAC, and the remainder digits are the CIN.

EXAMPLE 1 The 7-digit GCP "4123456" is encoded as DAID "4 123456" with IAC "4" and CIN "123456".

NOTE 1 Rare duplicate encodings can occur where the GCP CIN portion contains leading zeros.

GCPs up to 12 digits shall be encoded as follows:

— The first byte shall be set to 0xFE, followed by

— the 40-bit number encoding of the GCP.

EXAMPLE 2 The 12-digit GCP "377956789012" is encoded as ":FE:57:FF:FA:EB:14".

7.6 DigSig certificate identifier (CID)

The CID is a 16-bit unsigned integer reference number to a DigSig certificate. The Domain Authority manages the CID and its associated DigSig certificate.

The tuple {DAID, CID} shall be unique within the DigSig certificate validity period.

A CID may be reused. Substantial time between the uses is recommended.

7.7 DigSig validity

The UTC time zone shall be used for all validity periods.

DigSigs are issued within an issuing period as specified by the range of the DigSig timestamp (see 8.5). DigSigs are typically required to be valid for a much longer period than the issuing period. A DigSig expires when its associated certificate expires (see 7.3). The DigSig is therefore valid from the date of signing to the expiry of its associated DigSig certificate as illustrated in Figure 4. The DigSig remains verifiable, though the associated DigSig certificate has expired.

NOTE 1 A generally accepted best-practice is to limit the issuing period to one year to ensure that a new key pair is generated annually. This reduces the exposure risk of a key compromise. However, a DigSig is typically valid for a much longer period, e.g. a DigSig for a work of art may need to be valid for 40 years. In contrast, a DigSig for medicine may need to be valid for only 2 years. A much stronger crypto algorithm, resulting in a much larger signature, is required for the art than the medicine. "Re-keying" and DigSig certificate "re-issuing" ensure efficient and effective authentication of items, which is illustrated in Figure 4.

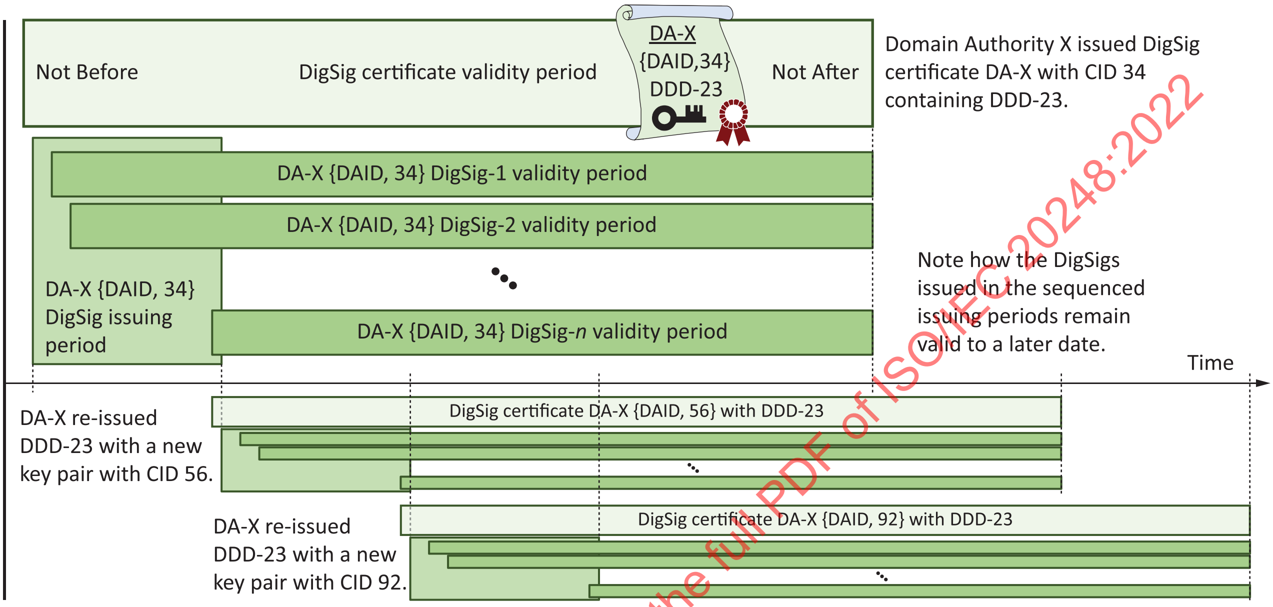

Not Before
DigSig certificate validity period
DA-X
{DAID,34}
DDD-23
Not After
Domain Authority X issued DigSig certificate DA-X with CID 34 containing DDD-23.
DA-X {DAID, 34} DigSig-1 validity period
DA-X {DAID, 34} DigSig-2 validity period
DA-X {DAID, 34} DigSig issuing period
DA-X {DAID, 34} DigSig-*n* validity period
Note how the DigSigs issued in the sequenced issuing periods remain valid to a later date.
Time
DA-X re-issued DDD-23 with a new key pair with CID 56.
DigSig certificate DA-X {DAID, 56} with DDD-23
DA-X re-issued DDD-23 with a new key pair with CID 92.
DigSig certificate DA-X {DAID, 92} with DDD-23

Figure 4 — DigSig validity period

NOTE 2 X.509 allows for the periodic re-issuing of a certificate to extend a certificate validity period.

EXAMPLE A re-issue period is discussed in the following use-case:

A university selects a one-year issuing period; they generate a new key pair annually and publish a new DigSig certificate with a new CID annually.

The university determines that the risk profile indicates that the DigSig certificate should be valid for 10 years. They therefore issue the DigSig certificate with a validity period of 10+1 years (to include the issuing period).

University certificates need to be verifiable for many years. The university therefore re-issues the DigSig certificate, with the same CID, DDD and key pairs, every decade. Both the DigSig and the current DigSig certificate remain therefore verifiable.

NOTE 3 Encryption strength is measured in the brute-force attack duration. The DigSig lifespan is limited by this duration. Re-issuing of the DigSig with new key pairs extends the lifespan of a DigSig.

7.8 DigSig certificate management

DigSig certificate management and DigSig certificate revocation shall comply with ISO/IEC 9594-1.

NOTE The revocation of a DigSig certificate revokes all DigSigs issued with it. The DigSigs need not be individually revoked.

7.9 DigSig revocation

The revocation of DigSigs is a recommended service typically performed by a trusted service (see Annex E).

The Domain Authority of a DigSig shall be the authority of the revocation of the DigSig.

The DDD may list additional authoritative revocation URIs (see 8.5).

The DigSig revocation shall comply with Annex J.

NOTE A chain of trusted digital signatures can be compromised in several possible ways (e.g. exposure of a private key, innovations in the mathematics of hashing algorithms, or availability of greater computing power). Any implementation of this document which does not implement revocation services for digital signatures assumes some risk of wrongly trusting a digital signature.

7.10 Online verification

Online verification, using commonly available browsers, may be performed by submitting the URI envelope (see 8.3.3.3) and a RAW envelope (8.3.3.2) prefixed with a verification URI to a trusted online service (see Annex E).

The DDD may list authoritative verification URIs (see 8.5).

The trusted online service shall perform the `entertext` pragma (see 9.2) through its user interface.

The trusted online service should use standard mobile app services to perform the other readmethod pragmas.

NOTE A primary goal of this document is to provide an offline method, whereby a reader/interrogator reads, decodes, and verifies data carrier data in an open and interoperable manner. The data read, and passed on to another application, as in IoT, remain verifiable.

8 DigSig Data Description (DDD)

8.1 General

The DDD is a hierarchical schema-based specification of the data fields contained or directly associated with an item, the DDD data. The DDD data are structured, signed, verified, encoded, decoded, stored, and read from data carriers as specified by the associated DDD. The encoded form of the DDD data is called a DigSig.

AIDC data carriers' data storage and data transfer are limited. This limitation should be considered when developing the DDD.

The DDD is a JSON construct. It has the following format:

```
{<digsiginfo>,<datafields>}
```

`digsiginfo` is a JSON object containing management definition fields.

`datafields` is a JSON array of objects each specifying a compulsory and application field.

The DDD is a tree structure. The leaves are the DDD data containers and the branches are the structuring of the containers.

Figure 5 illustrates all the components of DigSig envelope as specified by a DigSig Data Description (DDD).

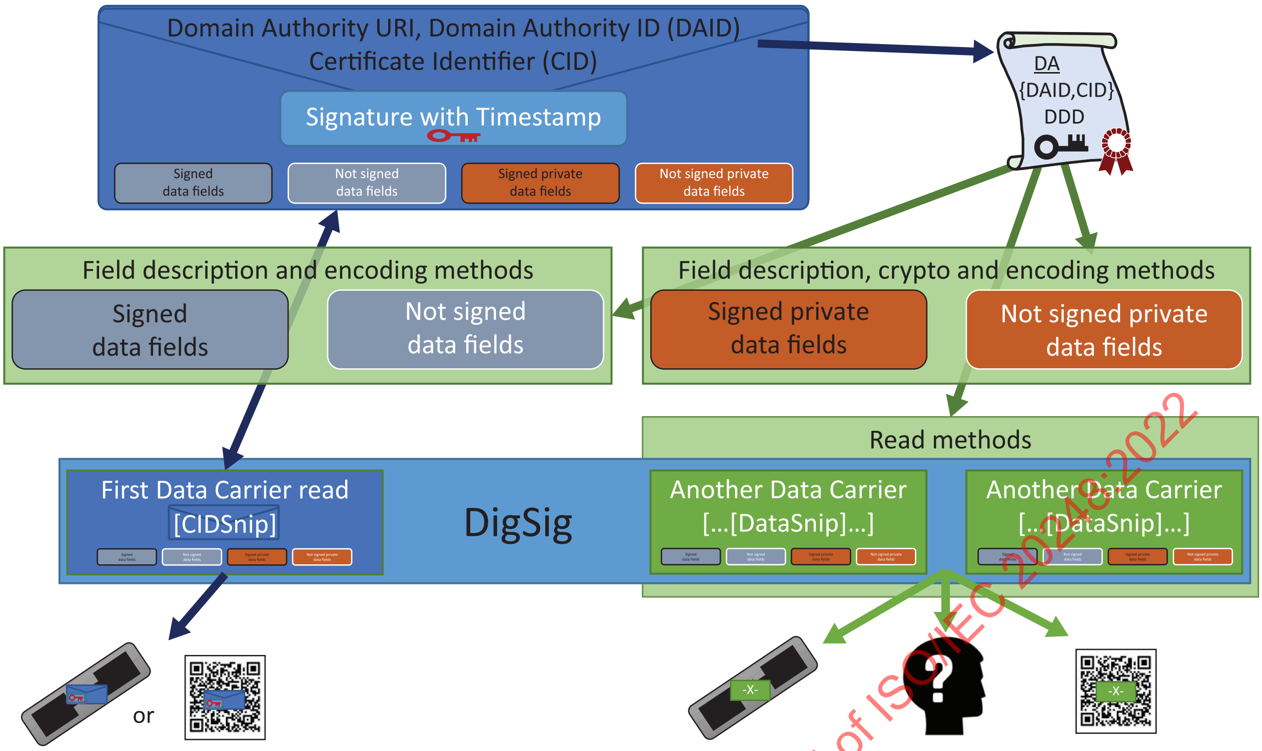

Domain Authority URI, Domain Authority ID (DAID)
Certificate Identifier (CID)
Signature with Timestamp
Signed data fields
Not signed data fields
Signed private data fields
Not signed private data fields
DA
{DAID,CID}
DDD
Field description and encoding methods
Signed data fields
Not signed data fields
Field description, crypto and encoding methods
Signed private data fields
Not signed private data fields
Read methods
First Data Carrier read
[CIDSnip]
DigSig
Another Data Carrier
[...[DataSnip]...]
Another Data Carrier
[...[DataSnip]...]
or
-X-
-X-

Figure 5 — Components of DigSig envelope as specified by a DigSig Data Description (DDD)

8.2 DDD derived data structures

8.2.1 General

DDD derived data structures are JSON constructs.

DDD derived data structures shall follow the DDD syntax as specified in 8.5.

The specified field order and hierarchy shall be maintained between a DDD and all its derived data structures.

EXAMPLE I.3 shows a DDD example.

8.2.2 DDDdata

DDDdata shall be a JSON array of arrays of values following the structure of the associated DDD `datafields` object (see 8.1), as such, DDDdata shall NOT contain JSON objects.

DDDdata shall only contain the values of the DDD leaf-fields, field values.

The JSON types for field values shall be:

- Single value fields (see 8.9.3.1 to 0): A JSON string, number, true, false, or null, e.g.

  `[…"abc","XX",123,true…]` the values of a string, an enumeration, a number and a boolean field.

- A structure defined by struct (see 8.9.3.9): A JSON sub-array, e.g.

  `[…["abc",123,true]…]` the values of a structure with a string, a number and a boolean field.

- An array of a field values, cardinality (see 8.9.2.2): a JSON array of the field values.

  `[…["abc","def"…]…]` the values of a list of a string field.

- An enumeration where the enumeration selects a set of fields, `enumvaluefields` (see 8.9.3.8): a JSON string followed by a JSON sub-array, e.g.

  `[…"XX",[123,true]…]` the values of the enumeration "XX" set of fields which is in this case a number and boolean field.

The values of the fields signature and timestamp of DDDdata^{input} shall be ignored.

NOTE DDDdata are used as an input and output structure: see in Figure 5 and 8.3.3.4 The values of the fields signature and timestamp are generated by the DigSig EncoderGenerator.

EXAMPLE For DDDdata^{input} see I.5. For DDDdata see I.6.

8.2.3 SigData

SigData shall be a JSON array, a subset of DDDdata.

SigData shall only contain the data fields specified to be signed and verified. SigData shall be generated by deleting all field values of which the field descriptor bsign equals false.

It is important that signed data are computed to be consistent, else a signature verification may be rejected even when the data is correct, but the formatting different. As such the following rules apply:

- The data to be signed is the full JSON array, starting with the first "[" and ending with the last "]".
- Empty fields and arrays shall be pruned.
- All whitespaces and formatting, outside of string values shall be removed.
- Numbers shall only contain the characters 0 to 9, the decimal point when the fraction is non-zero and the leading negative sign. Numbers shall not contain leading zeros and fraction trailing zeros.
- Base64 shall be in the format when encoded from the binary; it shall contain the padding characters.

NOTE The above rules are based on the output of typical JSON message generation methods supported by most common coding languages.

EXAMPLE See I.7.

8.2.4 DDDdataTagged

DDDdataTagged shall be a JSON object whereby each array and field value of DDDdata is tagged with its fieldid value as set in the DDD to become a JSON pair.

EXAMPLE 1 This example is abbreviated and edited for readability purposes. It has two single value fields (A and B), a field with cardinality (a list C with 2 values), an simple enumeration (E), an enumeration with value fields (for F1: F11 and F12, and F2: F21 and F22) and a structure (S) with two single value fields (S1 and S2).

DDD datafields:

```
[{"fieldid":"idA"},                                  # A single value
 {"fieldid":"idB"},                                  # A single value
 {"fieldid":"idC","cardinality":"{2}"},              # A list
 {"fieldid":"idE","enumvalues":["E1","E2"]},         # An enumeration
 {"fieldid":"idF","enumvalues":["F1","F2"],          # An enumeration
  "enumvaluefields":[[{"fieldid":"idF11"},{"fieldid":"idF12"}],      #  with fields
                     [{"fieldid":"idF21"},{"fieldid":"idF22"}]]},    #
 {"fieldid":"idS","fields":[{"fieldid":"idS1"},{"fieldid":"idS2"}]}] # A structure
```

DDDdata:

```
["valA","valB",["valC1","valC2"],"E1","F1",["valF11","valF12"],["valS1","valS2"]]
DDDdataTagged:
{"idA":"valA","idB":"valB",                        # Single values
 "idC":["valC1","valC2"],                          # A list
 "idE":"E1",                                       # An enumeration
 "idF":"F1","F1":{"F11":"valF11","F12":"valF12"},  # An enumeration with fields
 "idD":{"idS1":"valS1","idS2":"valS2"}}            # A structure
```

EXAMPLE 2 See I.8.

8.2.5 DDDdataDisplay

DDDdataDisplay is used to display DDDdata in the desired language and time zone. DDDdataDisplay shall only contain information for display purposes. All other information shall be purged.

See 8.10 for details.

EXAMPLE The following is the DDDdataDisplay `datafields` value in an abbreviated form for the example in 8.2.4.

```
[{"fieldid":"idA","displayvalue":"valA"},
 {"fieldid":"idB","displayvalue":"valB"},
 {"fieldid":"idC","displayvalue":["valC1","valC2"]},
 {"fieldid":"idE","displayvalue":"E1"},
 {"fieldid":"idF","displayvalue":"F1"},
 {"fieldid":"F1","displayvalue":[{"fieldid":"F11","displayvalue":"valF11"},
                                 {"fieldid":"F12","displayvalue":"valF12"}]},
 {"fieldid":"idD","displayvalue":[{"fieldid":"idS1","displayvalue":"valS1"},
                                  {"fieldid":"idS2","displayvalue":"valS2"}]}]
```

8.3 DigSig format

8.3.1 General

See Figure 5. The encoded form of the data of a DDD data is called a DigSig. A DigSig may be stored over more than one data carrier, including human memory and observation. It contains fields which are signed or not signed and open or private.

The DigSig envelope is the encapsulation of the DigSig. The first part of DigSig envelope shall be the CIDSnip with the first two fields the {DAID, CID}. The {DAID, CID} references the DigSig Certificate, which contains the DDD, which in turn defines methods to read, decode and verify data of the DigSig.

8.3.2 Snips

8.3.2.1 Snip types

A Snip is a continuous binary or text sequence of encoded field values.

The following Snips shall be used:

- CIDSnip: The CIDSnip is the head, the first Snip, of the DigSig. It shall contain the DAID, CID as the first two fields, and all data fields without a `readmethod` or `entertext` pragma.
- DataSnips: DataSnips are stored in another part of the CIDSnip data carrier, or on another data carrier as specified by the `readmethod pragma` (see 9.4). The `entertext pragma` (see 9.2) provides a method for a human to be the data carrier (memory and observation) of a DataSnip.

NOTE A Data Carrier will typically provide a CIDSnip in an automated manner, e.g. when reading a barcode, the content of the barcode will be the CIDSnip as a singular read. In contrast, inventory of an ISO/IEC 18000-63 will only provide the UII. It is possible for the UII to contain the complete DigSig, though it is expected that the TID and some user memory can also be read. In the latter case, the TID and user memory are provided as separate DataSnips.

8.3.2.2 Snip encoding

A Snip shall be encoded in binary using network byte order (see 4.4) using the DigSig Data Description (DDD).

A DataSnip may be provided as text (see 9.2 and 9.4). It shall be validated and converted to the DDD specified format.

DataSnips shall be identified by the `fieldid` value, the field short name, of the field contained by the DataSnip (see 8.8, 9.2, 9.4 and D.3.9).

The following shall apply to padding a Snip:

- The head of a Snip shall NOT be padded.
- The tail of a Snip may be padded with any bit value.

NOTE The DigSig decoder knows exactly the encoding format of the Snip, as such, padding is inherently ignored.

EXAMPLE See I.4.

8.3.3 Envelope format

8.3.3.1 General

The envelope contains fields as specified in 8.7.

8.3.3.2 RAW envelope

A RAW envelope is the CIDSnip.

The DataSnips are included in the envelope by reference (see 9.2 and 9.4) as specified by the DDD.

8.3.3.3 URI envelope

The URI envelope shall be the CIDSnip encoded in base64url preceded with a URI in accordance with IETF RFC 3986. The DataSnips are included in the envelope by reference (see 9.2 and 9.4) as specified by the DDD.

The URI envelope shall have the following format:

```
URI envelope |= <verificationuri> ‖ "?" ‖ Base64url_encoding (CIDSnip)
```

The verification URI used should be one of the listed `verificationuri`s (see 8.5).

NOTE The URI envelope is useful for online verification where a data carrier is read using a handheld device with generic data carrier readers, e.g. a smart phone with a barcode and/or RFID reader.

8.3.3.4 GS1 envelope

The GS1 envelope shall comply with the GS1 digital signature directives.

8.3.4 AIDC specific construction of a DigSig

A DigSig envelope (a formatted CIDSnip, see 8.3.3.3) may be stored on all AIDC data carriers capable of carrying binary data or a Base64 URL. The following methods apply:

- The application knows by design it is reading a DigSig from the AIDC data carrier. No special data carrier encoding is required to indicate the carried DigSig.
- An AIDC data carrier that is specified to carry a URL shall use the URI envelope (see Annex L).

— An AIDC data carrier that is specified to carry a DigSig:

  — For example, the ISO/IEC 15961-2 data construct register has assigned AFI 0x92 and DSFID 0x11 to indicate a DigSig (see Annex M).

  — For example, an ISO/IEC 15434-based message, having the ANSI MH10.8.2 Data Identifier "6R" (see Annex K) as part of the message, indicates a DigSig.

  The DAID encoding may be replaced with the 8-bit encoding 0xFF (reference DAID) where the AIDC data message contains an unambiguous DAID as the first part of the message (see M.4 for an ISO/IEC 17367 example).

The optional DataSnips may be stored on all AIDC data carriers capable of carrying binary or text, since the DDD specifies how a DataSnip is read.

An AIDC text-based data message stored in a DataSnips shall be pruned from all control and whitespace characters for signature generation and signature verification.

The construction and mapping of a DigSig in an AIDC data carrier is illustrated in Figure 6.

The data carrier should have sufficient carrying and transfer capacity for the specified Snip.

NOTE 1 A DigSig using a null signature (see B.4.4) can be as small as 32 bits, however, the smallest "applicable" signature is 256 bits resulting in a minimum DigSig size of 320 bits (see B.3.1). Using a DataSnip for the 256-bit signature can assist in reducing the impact of the carrier capacity and read limitations. It also provides a method to seamlessly introduce authenticity in legacy systems.

NOTE 2 Removing all control and whitespace characters from the text-based message eliminates any ambiguities that they can bring during the coding process.

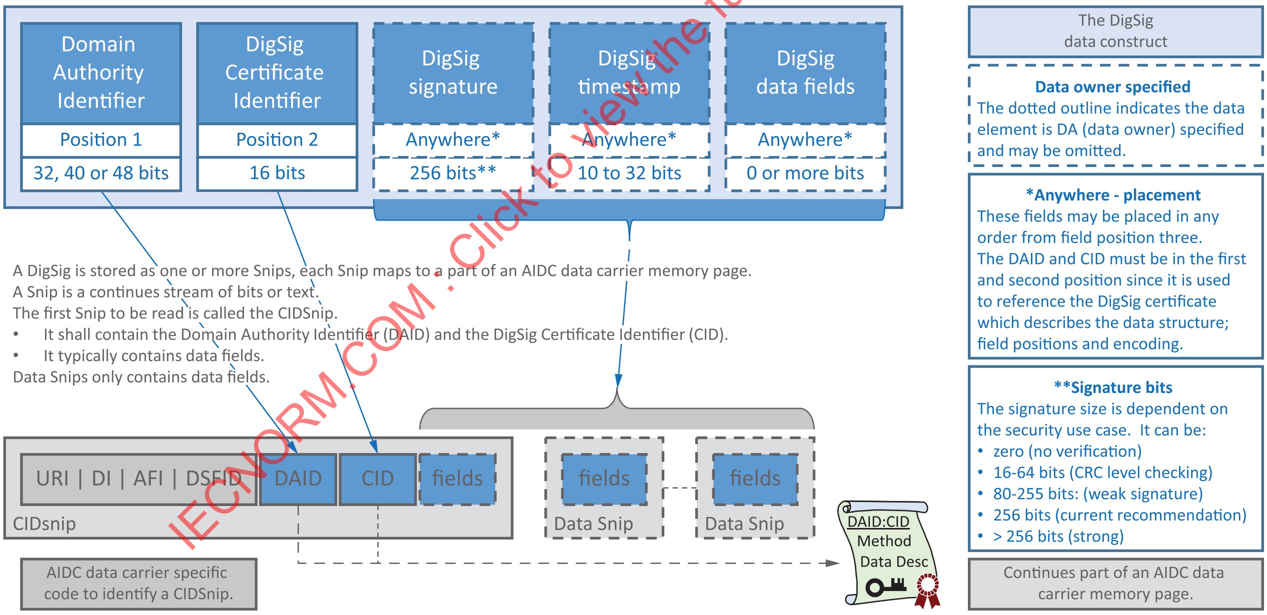

Domain Authority Identifier
Position 1
32, 40 or 48 bits
DigSig Certificate Identifier
Position 2
16 bits
DigSig signature
Anywhere*
256 bits**
DigSig timestamp
Anywhere*
10 to 32 bits
DigSig data fields
Anywhere*
0 or more bits
The DigSig data construct
Data owner specified
The dotted outline indicates the data element is DA (data owner) specified and may be omitted.
*Anywhere - placement
These fields may be placed in any order from field position three.
The DAID and CID must be in the first and second position since it is used to reference the DigSig certificate which describes the data structure; field positions and encoding.
**Signature bits
The signature size is dependent on the security use case. It can be:
• zero (no verification)
• 16-64 bits (CRC level checking)
• 80-255 bits: (weak signature)
• 256 bits (current recommendation)
• > 256 bits (strong)
A DigSig is stored as one or more Snips, each Snip maps to a part of an AIDC data carrier memory page.
A Snip is a continues stream of bits or text.
The first Snip to be read is called the CIDSnip.
• It shall contain the Domain Authority Identifier (DAID) and the DigSig Certificate Identifier (CID).
• It typically contains data fields.
Data Snips only contains data fields.
URI | DI | AFI | DSFID
DAID
CID
fields
CIDsnip
fields
Data Snip
fields
Data Snip
DAID:CID
Method
Data Desc
AIDC data carrier specific code to identify a CIDSnip.
Continues part of an AIDC data carrier memory page.

Figure 6 — The construction and mapping of an DigSig in an AIDC data carrier

8.4 The DigSig physical data path

The physical data path of a DigSig is illustrated in the steps below as depicted in Figure 7, and illustrated in Annex N.

$\mathtt{DDDdata}^{\mathtt{input}} \Leftarrow \mathtt{LocalApplicationGenerateDDDdata(Certificate}^{\mathtt{\{DAID,CID\}}}\mathtt{,ApplicationData)}$

$\mathtt{SigData} \Leftarrow \mathtt{ExtractSigData(Certificate}^{\mathtt{\{DAID,CID\}}}\mathtt{,\ DDDdata}^{\mathtt{input}}\mathtt{)}$

```
{Signature, Timestamp } ⇐ Sign(Certificate^{DAID,CID}, SigData)

DigSig ⇐ EncodeDigSig(Certificate^{DAID,CID}, DDDdata^{input}, {Signature, Timestamp})

Programmed data carrier… ⇐ WriteDigSig(DigSig)

{CID', DigSig'} ⇐ ReadDigSig(Data Carrier…)

{DDDdata’, SigData', Signature'} ⇐ DecodeDigSigCertificate^{DAID,CID}', DigSig')

VerificationOutcome ⇐ Verify(Certificate^{DAID,CID}', SigData’, Signature'}

ApplicationData ⇐ TranslateFromLocalApplication(Certificate^{DAID,CID}', DDDdata',

VerificationOutcome)
```

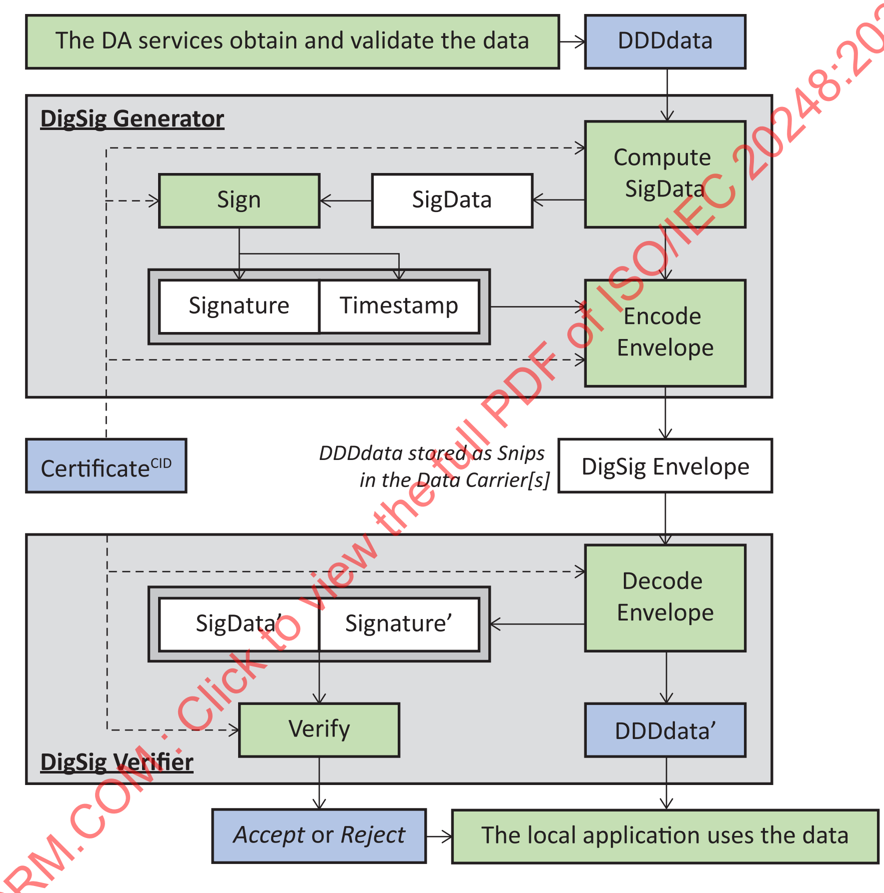


Figure 7 — Physical data path

8.5 DDD syntax

The DDD shall have the following syntax:

```
{"digsiginfo":
  {"specificationversion":<specificationversionvalue>,        # See 5.1.
   "dauri":<dauri>,
   "daid":<DAID>,                                             # See 7.5.
   "cid":<CID>,                                               # See 7.6.
   "verificationuri":[<verification URI>,…],                  # See 7.10.
   "revocationuri":[<revocation URI>,…],                      # See 7.9.
   "preverify":<display string before verification>,
   "acceptverify":<display string on verification accept>,
   "rejectverify":<display string on verification reject>,
   "postverify":<display string after verification>,
   "structureddocuri":{<language code>:<document URI>,…},     # See 8.11.
   "custom":<JSON value with custom information>},
"datafields":
 [{"fieldid":"specificationversion",     # Shall be the first field.
   "type":"string",
   "bsign":true},
  {"fieldid":"dauri",                    # Shall be the second field.
   "type":"string",
   "bsign":true},
  {"fieldid":"daid",                     # Shall be the third field (see 7.5).
   "type":"string",
   "bsign":true},
  {"fieldid":"cid",                      # Shall be the fourth field (see 7.6).
   "type":"unsignedint",
   "bsign":true},
  {"fieldid":"signature",                # This field may be placed elsewhere.
   "type":"bstring",                     # The elected signature method length applies
   "bsign":false,                        # as a fixed length (see B.4 for examples).
   "binaryformat":"{<signature length>}"},
  {"fieldid":"timestamp",                # This field may be placed elsewhere.
   "type":"isodate",                     # The time zone shall be UTC. range is used
   "bsign":true,                         # to configure the timestamp span to be at least
   "range":<DigSig issuing period>},     # as long as the issuing period (see 7.7).
  …]}                                    # Optional data fields.
```

specificationversion, dauri, cid, signature and timestamp are compulsory data fields. The compulsory fields shall only use the field descriptors as specified in this clause. The timestamp range field descriptor shall be included in the DDD. The other field descriptors may be omitted from the DDD.

EXAMPLE See B.3.1.

8.6 DigSig information fields

The DigSig information fields describe the use environment of the DigSig and DDD. digsiginfo fields are not included in the DigSig. digsiginfo fields, as part of the DDD, are included in the DigSig certificate, as such, it is a verifiable set of information applicable to all the DigSigs associated with a specific DigSig certificate.

NOTE DigSig information fields and the descriptions of the DigSig data fields are applicable to all the DigSigs generated using the DDD authenticated by the DigSig certificate. These "static" fields and information are not encoded on the AIDC data carrier with an explicit aim to keep the encoded bit count as low as possible. The AIDC data carrier is therefore only encoded with dynamic data (see 8.7).

digsiginfo is a JSON object. It shall contain only the fields specified in Table 5. These fields shall not contain DDDdata values.

Table 5 — digsiginfo fields

Name	Purpose and value	Type	Inclusion
specificationversion	Set according to 5.1.	JSON string	compulsory
dauri	The Domain Authority URI (see 6.2).	JSON string	compulsory
daid	The Domain Authority identifier (see 7.5).	JSON string	compulsory
cid	The DigSig certificate identifier (see 7.6).	unsigned integer	compulsory
verificationuri	These URIs point to the online verification service. It is also used for the URI envelope.	JSON array of strings	compulsory to list at least one
revocationuri	These URIs point to a revocation service.	JSON array of strings	compulsory to list at least one
preverify	The text shall be displayed before verification.	displaystring	optional
acceptverify	The text shall be displayed on a successful verification.	displaystring	optional
rejectverify	The text shall be displayed on a rejected verification.	displaystring	optional
postverify	The text shall be displayed after verification.	displaystring	optional

Table 5 *(continued)*

Name	Purpose and value	Type	Inclusion
structureddocuri	This URI shall point to a Media Type (IEFT RFC 6838) resource which is used by a direct substitution method to create a populated document reflecting the verified DigSig display data (see 8.11).	A JOSN object of JSON objects	optional
custom	JSON object with DDD custom information.	JSON object	optional

8.7 Data fields

8.7.1 General

The DDD datafields field specifies the information, structure, read method and encoding of the DDDdata fields. Only the values of the datafields are included in a DigSig. The static data is contained in the DigSig Certificate (see 8.6).

— `datafields` is a JSON array of objects.

— `datafields` shall contain the compulsory data fields as depicted in Table 6.

— `datafields` may contain application data fields.

8.7.2 Compulsory data fields

The compulsory DigSig data fields are specified in Table 6.

Table 6 — Compulsory DDD data fields

Names	DDD Type (type)	Signed (bsign)	DDDdata Order	DigSig Order
specificationversion	string	Yes	First	Not included
dauri	string	Yes	Second	Not included
daid	DAID rules (see 7.5)	Yes	Third	First
cid	16-bit unsignedint	Yes	Fourth	Second
signature	fixed length bstring	No	Default fifth	Default third
timestamp	isodate	Yes	Default sixth	Default fourth

The following special rules shall apply:

— Compulsory data fields should not contain field descriptors. The settings of Table 6 shall be used.

  — timestamp shall contain `range`.

  — signature may contain `binaryformat` to specify they signature length for DDD encoding analysis. The `binaryformat` shall be ignored. The signature length shall be set according to the signature method used.

— signature shall be generated at the point of signing using the DigSig certificate specified signature algorithm and its specified bit length.

— timestamp shall be generated at the point of signing using the specified range and accuracy (see 8.9.3.7.3).

8.7.3 Application data fields

Application data fields are specified by the domain authority for a specific use case. Application data fields are optional.

Application data fields may be included by codebook reference (see 8.11).

8.8 Data field object syntax

Data fields are JSON objects with the following syntax:

```
{"fieldid":<field short name>,
 "type":<DDD type>,
 <optional field descriptors>}
```

A data field object may contain field descriptors as specified in Table 7. These field descriptors provide a method to specify the field syntax, semantics and encoding. The semantics use the multi-language displaystring (see 8.10.2).

Table 7 — DDD field descriptors

Name	Purpose and value	Default	Type	Inclusion
fieldid	The short name of the field used as its identifier. The fieldid shall be unique within the DDD and shall only contain [A-Za-z0-9].	-	JSON string	compulsory
type	Specify the field type of application fields as boolean, unsignedint, number, string, bstring, digsigenv, isodate, enum, or struct. The type of the compulsory fields is specified in 8.7.2 and should be omitted.	-	JSON string	compulsory
fieldname	A display string that is intended to name the field differently from fieldid. This is for display purposes on capturing, printing, or extracting the data.	the value of fieldid	displaystring	optional
description	A display string to provide meaning to a field when capturing, printing, or extracting the data.	null	displaystring	optional
pragma	See Clause 9.	null	JSON object	optional
bsign	<true | false>; if true then the field value is included in the signature.	true	boolean	optional
cardinality	Specify a list of values for a field (see 8.9.2.2).	"{1}"	JSON string	optional
nullable	<true | false>; if true then the field value may be set to null (see 8.9.2.1).	false	boolean	optional
nulldescription	A display string to describe a nullable field when it contains a null.	null	displaystring	optional
nonnulldescription	A display string to describe a nullable field when it contains a value.	null	displaystring	optional
binaryformat	This field descriptor specifies the value format to achieve optimal binary encoding (see 8.9.3.x).	as specified by type	JSON string or number as specified by type	optional
range	The allowed value range of the field (see 8.9.3.x).	as specified by type	as specified by type	optional
decimalunit	Specifies the accuracy of a number (see 8.9.3.3).	0	integer	optional with the type number
truncate	Specifies the truncation of the input value of a number (see 8.9.3.3).	"round"	string	
timezone	Specify the time zone to be used with DDDdataDisplay (see 8.9.3.7 and 8.10.3.6).	"UTC"	JSON string	

Table 7 *(continued)*

Name	Purpose and value	Default	Type	Inclusion
enumvalues	Enumeration values (see 8.9.3.8).	null	JSON array of JSON strings	used with the type enum
enumvaluedesc	The description of the enumeration values (see 8.9.3.8).	null	JSON array of displaystring	optional with the type enum
enumvaluefields	A set of fields associated with the enumeration value (see 8.9.3.8).	null	JSON array of objects	
fields	A JSON object that contains the fields of the type struct (see 8.9.3.9).	at least one field	As specified by struct	used with the type struct
defaultvalue	The default value for the field when capturing the data.	null	As specified by type	optional
displayformat	The string format for DDDdataDisplay of the field value (see 8.10).	according to the type	As specified by type	optional
custom	JSON object with DDD custom information. Components of this standard shall ignore this field.	null	JSON object	optional

8.9 DDD field types and associate settings

8.9.1 General

JSON supports the following types: number, string and boolean, and the special value null. This document specifies rules for their use.

This document specifies the following additional DDD types: `unsignedint`, `bstring` and `isodate`. The JSON type for unsignedint and date shall be number and bstring shall be string.

This document further specifies complex field types `enum` and `struct`.

- `enum` maps to a JSON string.
- `enumvaluefields` maps to a JSON array of object containing the fields of the specific `enum` value.
- `struct` maps to a JSON array of object containing the fields of the `struct`.

8.9.2 Special field values

8.9.2.1 The DDD field value "null"

The value null is a special value, meaning "no value".

The DDDdata value shall be the JSON literal null if the field is set to null.

A field shall only take the value null if `nullable` is set to true.

The default is false.

A field which is nullable shall have the null bit in the MSB position, the first bit of the binary encoding of the field.

- If the null bit is set to false (1_2) then the field contains a value.
- If the null bit is set to true (0_2) then the field contains no value. The binary encoding of the field shall be collapsed.

If `nullable` is set to true; then, if set, the value of `nulldescription` or `nonnulldescription` shall be displayed within DDDdataDisplay.

See 9.4 for the null binary encoding where the readmethod pragma is used.

8.9.2.2 Array of field values - cardinality

`"cardinality":<field cardinality>`

The type cardinality specifies multiple field values for a field, a list of values.

The DDDdata shall be a JSON array of the field values.

The value of cardinality shall be a JSON string:

— `"{m}"` with m > 0 specifies a fixed number of values. The binary encoding shall reserve m fields.

EXAMPLE 1 "binaryformat":"{5}"

— `"{m,n}"` with 0 ≤ m < n specifies variable length list with m to n values. The encoded value count shall be relative to m, 0 to n-m (n-m+1 values).

EXAMPLE 2 "binaryformat":"{5,10}"

The binary encoding of a variable cardinality field shall be prefixed with `countbits`. `countbits` shall be encoded as follows:

```
countbits |= {<countsize>‖<fieldcount>}
```

countsize is a variable length bit field terminating with a 0_2, with each preceding 1_2 representing a nibble in `fieldcount`.

fieldcount is the number of field values. `fieldcount` is encoded as a multiple of nibbles ($CCCC_2$).

EXAMPLE 3 countsize 0_2 specifies m field values.

EXAMPLE 4 countsize 10_2 $CCCC_2$ specifies m plus 0 to 15 field values.

EXAMPLE 5 **countsize 110_2 $CCCC_2$** $CCCC_2$ specifies m plus 0 to 255 field values.

8.9.3 Field types

8.9.3.1 boolean

`"type":"boolean"`

A boolean field is a binary switch.

The DDDdata value shall be the JSON literals true, false.

EXAMPLE A DDDdataTagged example "IamSMART":true

The binary encoding of a boolean field shall be a single bit set to 0_2 when false, and 1_2 when true.

NOTE If the boolean field is marked as nullable, then the encoding is two bits long when not null (false: 10_2; true: 11_2) and one bit when null (0_2).

The following Table 7 field descriptors are valid for this field: `fieldname`, `description`, `pragma`, `bsign`, `cardinality`, `nullable`, `nulldescription`, `nonnulldescription`, `defaultvalue`, `displayformat` and `custom`.

8.9.3.2 unsignedint

`"type":"unsignedint"`

An unsignedint field shall contain a non-negative integer.

The DDDdata value shall be a non-negative JSON integer number in the decimal notation. Leading zeros and all formatting shall be stripped.

EXAMPLE 1 A DDDdataTagged example "appels":12345

NOTE unsignedint is a special implementation of `number`.

The binary encoding may be controlled with binaryformat and range.

- `binaryformat` may be used to set the number of bits to encode the value: $0 \leq \text{value} < 2^{\text{bits}}$. The default is 16 bits.

  EXAMPLE 2 `"binaryformat":5` resulting in a value range of 0 to 31.

- `range` may be used to define an inclusive field value range n to m with $0 \leq m < n$. `range` shall be specified with the inclusive block notation `"[m..n]"`. The encoded value shall be relative to m resulting in n – m + 1 encoding values, 0 to n - m. The binary encoding shall allocate $\text{RoundUp}(\text{LOG}_2(n - m + 1))$ bits for the relative field value. `binaryformat` shall be ignored when `range` is specified.

  EXAMPLE 3 `"range":"[200..400]"` resulting in an encoding for 201 values.

The following Table 7 field descriptors are valid for this field: `fieldname`, `description`, `pragma`, `bsign`, `cardinality`, `nullable`, `nulldescription`, `nonnulldescription`, `binaryformat`, `range`, `defaultvalue`, `displayformat` and `custom`.

8.9.3.3 number

`"type":"number"`

A number field shall contain a decimal number accurate to a decimal unit $10^{\text{decimalunit}}$, i.e. $X \times 10^{\text{decimalunit}}$.

The DDDdata value shall be a JSON number in the decimal notation. Leading and trailing fraction zeros, and all formatting shall be stripped.

EXAMPLE 1 A DDDdataTagged example "`Amount`":`12300`

EXAMPLE 2 A DDDdataTagged example "`Amount`":`123.32`

NOTE 1 A JSON number cannot indicate accuracy, e.g. 12300 is accurate to any `decimalunit` ≤ 2.

`decimalunit`, an integer, shall be used to set the decimal accuracy of the number to $10^{\text{decimalunit}}$. The default is zero.

EXAMPLE 3 For `"decimalunit":2`, the numbers are multiples of 100s, e.g. 12300, 100, -100 and 4560000.

EXAMPLE 4 For `"decimalunit":-2`, the numbers are multiples of 100ths, e.g. 1042.34, -0.25, 0.00 and -13242.12.

The value to be encoded shall be the integer counting part (X) of the number $X \times 10^{\text{decimalunit}}$ relative to the first value.

The binary encoding may be controlled with `binaryformat` and `range`.

- `binaryformat` may be used to set the number of bits to encode the value in the inclusive value range $(-2^{\text{bits-1}}) \times 10^{\text{decimalunit}}$ to $(2^{\text{bits -1}} - 1) \times 10^{\text{decimalunit}}$ in intervals of $10^{\text{decimalunit}}$. The default is 16 bits.

  EXAMPLE 5 `"binaryformat":16` with `"decimalunit":0`, the defaults, results in the default number range -32768 to 32767 encoded with the values 0 to 65535.

  EXAMPLE 6 `"binaryformat":16` with `"decimalunit":2` results in the number range -3276800 to 3276700 encoded with the values 0 to 65535.

EXAMPLE 7 "binaryformat":16 with "decimalunit":-2, the defaults, results in the number range -327.68 to 327.67 encoded with the values 0 to 65535.

— range may be used to define an inclusive field value range m to n using the inclusive block notation "[m..n]" with m < n where m = X × $10^{\textbf{decimalunit}}$ and n = Y × $10^{\textbf{decimalunit}}$ with X and Y integers used for encoding. The encoded value shall be relative to X resulting in Y – X + 1 encoding values, 0 to Y - X. The binary encoding shall allocate RoundUp(LOG_2(Y – X + 1)) bits for the relative field value. binaryformat shall be ignored if range is specified.

EXAMPLE 8 "range":[0..10] with "decimalunit":-2, will be encoded in 10 bits with the values 0 to 1001. 10 bits can represent 1024 values, as such the last 23 encodable values are not valid.

EXAMPLE 9 "range":[-10..10] with "decimalunit":-2, will be encoded in 11 bits with the values 0 to 2001. 11 bits can represent 2048 values.

NOTE 2 unsignedint is a special case number where range starts with a zero and decimalunit is set to 0.

truncate may be used to specify the input value rounding to the accuracy of $10^{\textbf{decimalunit}}$. The value of truncate shall be a JSON string:

ceiling |= round up; add $10^{\text{decimalunit}}$ and truncate the fraction

round |= round to nearest with tie breaking towards ceiling

floor |= round down; truncate the fraction

round is the default.

EXAMPLE 10 The input value 123.45 will be ceiling truncated as follows: for decimalunit = -1 the value becomes 123.5 and for decimalunit = 1 the value becomes 130. This is useful for an automated machine interface, e.g. an automated scale providing not-formattable values. It is important to note that the scale may provide higher accuracy values than the application requires. Such high accuracy values may be problematic for signature verification.

The following Table 7 field descriptors are valid for this field: fieldname, description, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription, binaryformat, range, decimalunit, truncate, defaultvalue, displayformat and custom.

8.9.3.4 string

"type":"string"

A string field contains a case sensitive Unicode string encoded using UTF-8.

The DDDdata value shall be a JSON string.

EXAMPLE 1 A DDDdataTagged example "Notice":"Please eat your apples."

EXAMPLE 2 A DDDdataTagged example "City":"東京"

binaryformat specifies the number of characters of the string value. If binaryformat is not specified, then the string value shall be of variable length.

The value of binaryformat shall be a JSON string:

— "{m}" with m > 0 specifies a fixed length string of m characters.

EXAMPLE 3 "binaryformat":"{5}"

— "{m,n}" with 0 ≤ m < n specifies a variable length string of m to n characters. The encoded string length shall be relative to m, 0 to n - m (n – m + 1 characters).

EXAMPLE 4 "binaryformat":"{5,10}"

The string shall be padded at the end with the first character of the range of characters to be used. This character has a binary encoding of zero. Too long input values shall be truncated at the end.

The binary encoding of a variable length string shall be prefixed with `lengthchars`. `lengthchars` shall be encoded as follows:

`lengthchars |= {<lengthsize>||<numberofchars>}`

> `lengthsize` is a variable length bit field terminating with a 0_2, with each preceding 1_2 representing a nibble in `numberofchars`.
>
> `numberofchars` is the number of characters in the string. `numberofchars` is encoded as a multiple of nibbles ($CCCC_2$).

EXAMPLE 5 `lengthchars` 0_2 specifies a string with m characters.

EXAMPLE 6 `lengthchars` 10_2 $CCCC_2$ specifies a string with m plus 0 to 15 characters.

EXAMPLE 7 `lengthchars` 110_2 $CCCC_2$ $CCCC_2$ specifies a string with m plus 0 to 255 characters.

The default is UTF-8.

NOTE 1 Using the default string definition can result in unpredictable data length even when specifying a fixed length string, given that a UTF-8 encoded character can be expressed in more than 1 byte – specifically where the string contains non-ASCII characters.

Range should be used in all cases.

`range` limits the characters set. The value of `range` shall be a JSON string using a subset of the POSIX BRE bracket expression "[ ]" to list (not match) a range of characters. The following rules applies:

- Characters shall be listed explicitly.
- Characters shall be listed only once. All characters and notations used for matching and repeating shall be treated as literals, e.g. "*" is a literal and does not indicate a repeat.
- The `\\u{XXXX}` Unicode character notation may be used to specify a character with X a capital hex character. This notation shall be treated as a literal string representing the character.

  NOTE 2 `\\u{XXXX}` is used not to be confused with the JSON `\uxxxx` character codepoint. The first backslash escape the second backslash with u{XXXX} normal text.

- "-" may be used to list a range of characters in the UTF-8 scalar value range. "-" is a literal when following a "[" or lead a "]".
- "[^]" is used to exclude characters. "^" is a literal when not following a "[".
- Allowed POSIX BRE classes are: `[:alpha:]`, `[:alnum:]`, `[:xdigit:]`, `[:print:] and [base64url]`.

See Table 8 for examples.

The binary encoding of each character shall be

- The UTF-8 code points in the default case, and
- an enumeration of the specified Graphic UTF-8 characters
- using the UTF-8 scalar value
- in the order of specification occurrence
- with the enumeration starting with the value zero
- encoding each character in the specified set in RoundUp(LOG$_2$(number of characters in the set)) bits.

— Where non-Graphic code points appears in a range the following applies (for ease of implementation):

  — The non-Graphic code point is included in the range.

  — The non-Graphic code point shall be ignored and have no effect.

  EXAMPLE 8 In an application that requires 6 characters to be limited to 0, 1, 2, 3, A and B, the bracket expression defines the range as [0-3AB]. The characters map as follows: "0"→0, "1"→1, "2"→2, "3"→3, "A"→4 and "B"→5 using 3 bits per character.

Table 8 — Binary encoding ranges

Range	Bits/char (chars)	Notes
[A-Za-z0-9] or [[:alnum:]]	6 (62)	Alphanumeric.
[0-9A-F]	4 (16)	Uppercase hex.
[NY]	1 (2)	A simple NO-YES flag when the sting length is set to 1.
[0-9.]	4 (11)	Numbers of the format 999.999.
[0-9A-Z]	6 (36)	Uppercase alphanumerical.
[0-9A-Z[^IOQU]]	5 (32)	32 easy recognizable human readable characters for licence plates (I and 1; 0, O and Q; and V and U are sometimes confused).
[ก-๛]	7 (91)	The Thai character set. Though the range is from u{0E01} to u{0E5B} it contains only 86 Graphic characters. The 5 "missing" characters shall be treated as non-Graphic.
[一-鿕]	15 (20949)	The CJK Unified Ideographs character set range is u{4E00} to u{9FD5}.
[[:alnum:]ก-๛]	8 (62+91)	The Thai character set and alphanumeric characters.
[\\u{0020}-\\u{007E}] or [[:print:]]	7 (96)	Unicode expression for space to tilde, the printable characters of US-ASCII.
[#\\u{0020}*A-Za-z]	6 (54)	Unicode expression for `[# *A-Za-z]`.

The following Table 7 field descriptors are valid for this field: `fieldname`, `description`, `pragma`, `bsign`, `cardinality`, `nullable`, `nulldescription`, `nonnulldescription`, `binaryformat`, `range`, `defaultvalue`, `displayformat` and `custom`.

8.9.3.5 bstring

`"type":"bstring"`

A bstring field contains binary data.

The DDDdata value shall be a `HexString` as specified in 4.4.

`binaryformat` specifies the number of bits for the bstring value. If `binaryformat` is not specified, then the bstring value shall be variable length set according to the DDDdata input value, a multiple of 4 bits.

The value of `binaryformat` shall be a JSON string:

— `"{m}"` with m ≥ 0 specifies a fixed number of m bits. The binary encoding shall reserve m bits.

  EXAMPLE 1 `"binaryformat":"{5}"`

  NOTE m = 0 is allowed to allow for the correct handling of a null-signature (see B.4.4).

— `"{m,n}"` with 0 ≤ m < n specifies variable length bstring with m to n bits. The encoded bit length shall be relative to m, 0 to n - m (n – m + 1 bits).

EXAMPLE 2 "binaryformat":"{5,10}"

0_2 shall be used for padding. Too long input values shall be truncated at the end.

The binary encoding of a variable length bstring shall be prefixed with lengthbits. lengthbits shall be encoded as follows:

lengthbits |= {<lengthsize>||<numberofbits>}

lengthsize is a variable length bit field terminating with a 0_2, with each preceding 1_2 representing a nibble in numberofbits.

numberofbits is the number of bits for the bstring. numberofbits is encoded as a multiple of nibbles (CCCC$_2$).

EXAMPLE 3 lengthbits 0_2 specifies a bstring with m bits.

EXAMPLE 4 lengthbits 10_2 CCCC$_2$ specifies a bstring with m plus 0 to 15 bits.

EXAMPLE 5 lengthbits 110_2 CCCC$_2$ CCCC$_2$ specifies a bstring with m plus 0 to 255 bits.

The following Table 7 field descriptors are valid for this field: fieldname, description, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription, binaryformat, defaultvalue, displayformat and custom.

8.9.3.6 digsigenv

"type":"digsigenv"

The digsigenv field is a special designated string indicating that it contains a DigSig envelope resulting in nested DigSigs.

The DDDdata value shall be a Base64String as specified in 4.4.

The binary encoding of digsigenv shall be string with a range of Base64String.

The DecoderVerifier shall decode and verify the nested DigSig envelope independently from the parent DigSig.

The following Table 7 field descriptors are valid for this field: fieldname, description, displayformat, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription and custom.

8.9.3.7 Datetime

8.9.3.7.1 General

The ISO/IEC 20248:2018 date type has been found to be limiting in range and readability. This document specifies isodate superseding date as the datetime type.

The compulsory field timestamp shall use isodate.

8.9.3.7.2 date (superseded by isodate)

"type":"date"

date shall not be used after "specificationversion":"ISO/IEC 20248:2018". date is included in this specification for referencing when updating and reissuing DDDs.

NOTE 1 The DigSig timestamp predating this document uses date. This document specifies isodate as the type for timestamp.

A date field contains an exact 32-bit POSIX epoch date as specified in ISO/IEC/IEEE 9945 or a date relative to `Certificate.Data.Validity.Not Before` of the DigSig Certificate.

NOTE 2 Any other desired date format is representable as a string or a struct with the structjoin pragma.

The DDDdata value shall be a JSON number.

The value of binaryformat shall be a JSON string with:

`tepoch` |= This is the default value. 32-bit POSIX epoch, display format: `"YYYY-MM-DDThh:mm"`

`t1dm` |= 20-bit delta minutes max. range 1 year, display format: `"YYYY-MM-DDThh:mm"`

`t2dd` |= 10-bit delta days max. range 2 years, display format: `"YYYY-MM-DD"`

`t5dh` |= 16-bit delta hours max. range 5 years, display format: `"YYYY-MM-DDThh:00"`

`t5dd` |= 12-bit delta days max. range 5 years, display format: `"YYYY-MM-DD"`

The time zone of date shall be specified with `timezone`. The value of `timezone` shall be a JSON string with:

`utc` |= This is the default value. The DDDdata shall use UTC. DDDdataDisplay shall append "UTC".

`device` |= The DDDdata shall use UTC. DDDdataDisplay shall translate DDDdata to the device time zone and append the device time zone.

`tag` |= The DDDdata shall be time zone agnostic. A relative date shall use `Certificate.Data.Validity.Not Before` in a time zone agnostic manner. DDDdataDisplay shall NOT append a time zone.

NOTE 3 DDDdataDisplay is generated by the DecoderVerifier.

`range` shall use the range block notation `"[m..n]"` to specify an inclusive date range m to n with m and n dates in `tepoch` numbers.

8.9.3.7.3 isodate

`"type":"isodate"`

An isodate field contains an ISO 8601 UTC datetime value accurate to milliseconds within a specified range and granularity. A range and granularity shall be specified.

Care should be taken that the range cater for the locality range of use.

NOTE 1 DDDdata does not cater for a time zone since it is specified to be UTC.

The DDDdata isodate value shall be a JSON string containing an ISO 8601 datetime day or week format to the specified accuracy.

— Day format: `YYYY[-MM[-DD[Thh[:mm[:ss[.s[s[s]]]]]]]]`.

— Week format: `YYYY-Www[-D[Thh[:mm[:ss[.s[s[s]]]]]]]`.

The characters "-", ":", "W" and "T' shall be used where applicable.

EXAMPLE 1 Day: "2009-07-12T03:02:45.123", "2009-07-12T03:02:45", "2009-07-12T03:02" and "2009-07"

EXAMPLE 2 Week: Monday 29 December 2008 is written "2009-W01-1" and Sunday 3 January 2010 is written "2009-W53-7".

The `isodate` binary encoding shall be the bit concatenation of each of the datetime components; year, month day, week, weekday, hour, minutes, seconds, and the specified tenth, hundredth or thousands of a second fraction. Table 9 provides the details for the datetime component encoding. Table 10 provides encoding examples.

The datetime range and granularity shall be specified with the inclusive range block notation `"[m..n]"` (`range` examples are provided in Table 9).

- m and n, with m before n, shall be datetime strings of the same type (day or week).
- The granularity shall be set by the granularity of m.
- The range ends with the latest datetime value of n within the granularity of m.
- m may use the plus character ("+") to indicate a date range using "+y[-m[-d]]" or "+y[-Ww].
  - m, d and w are optional.
  - y and m may be set to zero.
- `range` may be set to `"[tnull]"` where `tnull` indicates "no date time" with a binary encoding of zero bits.

Table 9 — `isodate` component encoding

Component	Values	Bits	Encoding method
Year (YYYY)	0000 to 9999	0 to 14	Indexed. The first year in the range shall be encoded as 0. Zero bits are used when the range contains only one year value, one bit for two values, to 14 bits for 9999 values.
Months (MM)	01 to 12	0 or 4	Encode the number value as is. January shall be encoded as 1.
Weeks (ww)	01 to 53	0 or 6	Encode the number value as is.
Days (DD)	01 to 31	0 or 5	Encode the number value as is.
Weekdays (D)	1 to 7	0 or 3	Encode the number value as is. Monday is the first day of the week encoded as 1.
Hours (hh)	00 to 23	0 or 5	Encode the number value as is.
Minutes (mm)	00 to 53	0 or 6	Encode the number value as is.
Seconds (ss)	00 to 53	0 or 6	Encode the number value as is.
Tens of a second (.s)	.0 to .9	0 or 4	Encode the number value as is. Mutually exclusive with .ss and .sss.
Hundreds of a second (.ss)	.00 to .99	0 or 7	Encode the number value as is. Mutually exclusive with .s and .sss.
Milliseconds (.sss)	.000 to .999	0 or 10	Encode the number value as is. Mutually exclusive with .s and .ss.

Table 10 — isodate encoding examples

Range	Encoded components	Last value	Year values
[2020..2021] [2020..+2]	YYYY	2021	2 → 1 bit
2020 encode as 0_2 and 2021 encode as 1_2 length 1 bit.			
[2020-01-01..2020] [2020-01-01..+1] [2020-01-01..+0-12	MM-DD	2020-12-31	1 → 0 bits
2020-08-16 encode as 1000_2 10000_2 length 9 bits.			
[2020-10-20T00:00:00..+2]	YYYY-MM-DDThh:mm:ss	2022-10-19T23:59:59	3 → 2 bits
2021-08-16T04:32:48 encode as 01_2 1000_2 10000_2 00100_2 100000_2 110000_2 length 26 bits.			

Table 10 *(continued)*

Range	Encoded components	Last value	Year values
[2020-10-20T00:00:00..+0-0-10]	DDThh:mm:ss	2020-10-29T23:59:59	1 → 0 bits
2020-10-25T00:00:00 encode as 11001_2 00000_2 000000_2 000000_2 length 22 bits.			
[2020-10-20T00:00:00.0..+1]	YYYY-MM-DDThh:mm:ss.s	2021-10-19T23:59:59.9	2 → 1 bit
2020-10-25T00:00:00 encode as 0_2 11001_2 00000_2 000000_2 000000_2 0000_2 length 27 bits.			
[2020-10-20T00:00:00.000..+1]	YYYY-MM-DDThh:mm:ss.sss	2021-10-19T23:59:59.999	2 → 1 bit
2020-10-25T00:00:00 encode as 0_2 11001_2 00000_2 000000_2 000000_2 0000000000_2 length 33 bits.			
[2020-W12-1T00:00:00..+1]	YYYY-Www-DThh:mm:ss	2021-W11-7T23:59:59	2 → 1 bits
2020-W32-4T04:32:48 encode as 0_2 10000_2 100_2 00100_2 100000_2 110000_2 length 27 bits.			
[2020-W12-1T00:00:00..+0-W27]	Www-DThh:mm:ss	2020-W39-7T23:59:59	1 → 0 bits
2020-W32-4T04:32:48 encode as 10000_2 100_2 00100_2 100000_2 110000_2 length 26 bits.			

The following Table 7 field descriptors are valid for this field: fieldname, description, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription, range, defaultvalue, displayformat and custom.

NOTE 2 Leap years and leap seconds are inherently catered for.

NOTE 3 The binary encoding size can be reduced by careful consideration of limiting the range.

NOTE 4 Any other desired datetime format, e.g. time only, can be constructed using struct with structjoin of unsignedint and number fields.

NOTE 5 Duration values can be specified using unsignedint or struct with structjoin of unsignedint and number fields.

8.9.3.8 enum

An enum is a single selection of a zero-based index of an enumeration of values in the order of specification. Each enumeration value may have a set of associated fields. The syntax of an enum field shall be as follows:

```
{"fieldid":<field short name>,
 "type":"enum",
 "enumvalues":<a JSON array of enum values>,
 "enumvaluedesc":<a JSON array of enum displaystring values>,
 "enumvaluefields":<a JSON array of arrays of fields associated with an enum value>,
 <optional field descriptors>}
```

EXAMPLE 1 See 8.2.4 and 8.2.5.

The enum descriptors are specified in Table 11.

Table 11 — enum field descriptors

Name	Purpose and value	Type	Inclusion
enumvalues	The enumeration values.	JSON array of JSON strings	compulsory
enumvaluedesc	The description of the enumeration values.	JSON array of displaystring	optional
enumvaluefields	A set of fields associated with the enumeration value.	JSON array of field objects	optional

The DDDdata value shall be a JSON string selected from enumvalues followed by the optional array of enumvaluefields values.

```
"<enum selection>",[<enumvaluefields value>…]
```

DDDdataTagged shall use the enum selection as the fieldname for the optional enumvaluefields array.

```
"<enum fieldid>":"<enum selection>","<enum selection>":{<enumvaluefields tagged fields>…}
```

EXAMPLE 2 "PlateType":"US","US":{"number":"ABC123","Chars":"white","Plate":"black"}

Care should be taken where enumvaluefields is used that the enum selection string is used as a JSON fieldname on the same level than the fieldid. Otherwise fieldid conflicts may be caused.

The binary encoding of an enum is the index value encoded with RoundUp(LOG$_2$(number of enumerations)) bits. The binary encoding of enumvaluefields shall follow the enum binary encoding in the order of DDD occurrence.

The following Table 7 field descriptors are valid for this field: fieldname, description, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription, enumvalues, enumvaluedesc, enumvaluefields, defaultvalue, displayformat, displayvalue, and custom.

8.9.3.9 struct

The struct type shall be used to group fields in a hierarchical manner. The value a struct field shall be an array of fields. The syntax of a struct field shall be as follows:

```
{"fieldid":<field short name>,
 "type":"struct",
 "fields":<a JSON array of fields>,
 <optional field descriptors>}
```

EXAMPLE See 8.2.4 and 8.2.5.

The DDDdata value shall be a JSON array of field values.

The bsign value of a struct shall be the default bsign value of the fields of the struct.

NOTE 1 Cardinality repeats the struct.

NOTE 2 A struct is nullable.

The following Table 7 field descriptors are valid for this field: fieldname, description, pragma, bsign, cardinality, nullable, nulldescription, nonnulldescription, fields, displayformat and custom.

8.10 DigSig data presentation

8.10.1 General

DDDdataDisplay presents a DigSig in a human readable format within a specified language and time zone.

DigSig display data comprises of:

— Locality processed displaystrings.

— Field values formatted as specified by displayformat.

DDDdataDisplay have the following format:

```
{"digsiginfo":{<DigSig information field>:<locality processed displaystring>,…},
 "datafields":
  [{"fieldid":<the fieldid value>,
    "fieldname":<the fieldname value>,
    "description":<a dash-concatenation of locality processed applicable descriptions>,
    "displayvalue":<the formatted field value as a string,
                    an array of struct and enumvaluefields in the datafields format, or
                    for cardinality, an array of displayvalues>}…]}
```

EXAMPLE See I.9.

8.10.2 displaystring

```
"displaystring":{<language tag>:<string to be displayed>,…}
```

`displaystring` specifies language strings with a JSON value object listing on one or more language strings. The language string fieldname specifies the language using the appropriate language tag. The language string value is a string in the specified language. The first element in the list is the default.

The language tag shall be constructed in accordance with IETF RFC 5646. The use of ISO 639-1, 2-character language tag is compulsory. The IETF RFC 5646 sub-tags are optional.

NOTE IETF RFC 5646 specifies that the language tag is case insensitive but recommends case use.

EXAMPLE 1

```
displaystring:{"en":"This is the display text",
               "af":"Hierdie is vertoonteks",
               "de-AT":"Dies ist der Anzeigetext "}
```

EXAMPLE 2 Language tags examples are, English: `"en":"January"`, Austrian German: `"de-AT":"Jänner"`, German: `"de":"Januar"`, Swiss German `"de-CH":"Januar"`, and Swiss French `"fr-CH":"Janvier"`.

8.10.3 displayformat

8.10.3.1 General

The displayformat field descriptor is type specific as specified in Table 12. The format is:

```
"displayformat":<type specific display format>
```

Table 12 — Display formats

Type	Display format
Boolean	A JSON object with the displaystring fields "true" and "false" (see 8.10.3.2).
Unsignedint	Number display format (see 8.10.3.3).
Number	Number display format (see 8.10.3.3).
String	None, display the DDDdata as is.
Bstring	Binary string display format (see 8.10.3.4).
Digsigenv	A displaystring using ${DAID}, ${CIDdec}, ${CIDhex}, ${CIDHEX} and ${env} (see 8.10.3.5).
Date	Convert the epoch date number to an ISO 8601 datetime format in the specified time zone. NOTE Included for backwards compatibility.
Isodate	Convert the date to the specified time zone (see 8.9.3.7.3).
Enum	A displaystring using ${enumvalues} and ${enumvaluedesc} (see 8.10.3.7).
Struct	Struct display format (see 8.10.3.8).

8.10.3.2 boolean display format

The boolean display format is a JSON object which specifies a `displaystring` for the true and false values, with `"en":"true"` and `"en":"false"` the defaults.

```
"displayformat":{"true":<true displaystring>,"false":<false displaystring>}
```

EXAMPLE

```
{"fieldid":"IamSMART",
 "type":"boolean",
 "displayformat":{"true":{"en":"Absolutely"},"false":{"en":"A pipe-dream"}}}
```

Field value: `true`

`{"fieldid":"IamSMART","displayvalue":"Absolutely"}`

8.10.3.3 number display format

The number display format is used with `number` and `unsignedint`. The number display format uses a mask-string to specify a decimal point number format (to be consistent with the JSON decimal format number).

- The mask-string shall use the mask-characters "9" to indicate a leading-space number and "0" to indicate a leading-zero number. The mask-characters shall not be mixed.
- The sign position shall use the mask-characters "-" to indicate the position of the negative sign and "+" to force both the plus and minus signs to be used.
  - The sign may only be placed as the first or last character of the display string or lead or trail the number. The plus and minus shall be used as literals in all other positions. In the following format "±" indicates valid positions in the mask-string.

    `"±<prefix>±<number  mask>±<postfix>±"` using the value -45.02 may result in `"-45.02"`, `" 45.02-"`,`"-$45.02"`,`"$ -45.02"`, `"$ 45.02-"`, `"0452"` and `"-45°C"`.
  - The sign-mask-character within a fixed length masked string shall be replaced with a space, a plus or a minus, e.g. `" 23"`, `"-23"` and `"+23"`.
  - The sign-mask-character within a variable length masked string shall add a plus or a minus in the position indicated, e.g. `"23"`, `"-23"` and `"+23"`.
- The default position for the sign shall be the first character of the string.
- `decimalunit` shall be used to determine the length of the fraction, which shall be zero padded.
- Variable length number mask:
  - A mask- string with a single character nine shall be used, e.g. `"9"`, `"$9"`, `"-$9"`, `"$-9"`, `"-9  %"`, `"+9°C"` and `"-AU$9"`.
  - The standard variable 3-digit grouping shall be indicated with the single character nine prefixed with a comma or a space indicating comma and space separation, e.g. `",9"`, `"-,9"`, `"-$,9"` with a decimal unit of -2, and `"+,9°C"` may result in `"12,345"`, `"-12,345"`, `"$12,345.00"` and `"+12,345°C"`.
  - The comma and space may be escaped, using `"\\"`, to treat it as a literal within the mask, e.g. `"BASH\\ 9"` will not be treated as a space-grouped number, but rather as variable length number prefixed with `"BASH "`. This example also illustrates that this mask is not in conflict with a fixed length mask where the number only contains a single digit, with the understanding that the fraction is specified by `decimalunit`.
- Fixed length number mask:
  - A format string with two and more nine characters and two or more zero characters shall be used, e.g. `"9999"` and `"-$99999.99"` and `"0000"`.
  - Zero-padding shall be used when the masked fraction is longer and `truncate` shall be used when it is shorter.
  - All other characters shall be treated as literals to separate, prefix, and postfix the number e.g. `"999-999"`, `"+99.99.99.99"`, `"00-000.00"` and `"EAN-13 0 00000 000000"`.
  - The mask-characters "0", "9", "+" and "-" shall be escaped, using `"\\"`, to use it as a literal in the mask-string, e.g. `"Code-\\07 9999-99"` may result in `"Code-07 1234-56"`.
- The default is "9" for `unsignedint` and "-9" for `number` indicating a variable length display string.

8.10.3.4 bstring display format

The bstring display format uses a mask-string to specify the binary formatting.

- `"Base64String"` shall be used to specify a variable width the base64 format as specified in 4.4.
- `"HexString"` shall be used to specify a variable width hex format as specified in 4.4.
- The mask-string shall use the mask-characters "0", "x" and "X" to indicate a binary, lowercase hex, and an uppercase hex character position in the mask-string. The mask-characters shall not be mixed.
- A single mask-character mask-string indicates a variable length binary display string.
- A single group of mask-characters lead or trailed by a single character, the separator, specifies a repeating grouping of the length of the group with a leading separator indicating the displaystring to start with the separator, e.g. `":XXXX"` may result in `":048A:8DE"` and `"XXXX "` will, with the same binary, results in `"048A 8DE"`. A hex character shall be completed with zero-bit padding.
- Two and more characters specify a fixed length mask.
  - A too long binary shall be displayed with a string of hashes ("#") matching the mask. A too short binary shall be padded with zero bits.
  - All characters, excluding "0", "x" and "X" shall be treated as literals to separate, prefix and postfix the binary string, e.g. `":xx:xx:xxxx"` specifies an eleven-character display string for a binary of up to 32 bits, and `"0000 00"` specifies a seven-character display string for a binary of up to 6 bits.

8.10.3.5 digsigenv display format

The digsigenv display format is a `displaystring` using the parameters `${DAID}`, `${CIDdec}`, `${CIDhex}`, `${CIDHEX}` and `${env}`, as specified in Table 13, with the default `"en":"${DAID} ${CIDdec}"`.

Table 13 — `digsigenv` display parameters

`digsigenv` parameter	**Description**
DAID	The DAID as specified in 7.5.
CIDdec	The CID as a decimal value.
CIDhex	The CID as a lower-case hex value xxxx.
CIDHEX	The CID as an upper-case hex value XXXX.
env	The DigSig envelope in Base64String.

The displayvalue should be displayed as a DigSig verification hyperlink by an DigSig verification user interface.

EXAMPLE The CID is not shown since it is not important.

```
"displayformat":{"en":"Click this link to verify the ${DAID} container ID."}
"displayvalue":"Click this link to verify the QC ABCD container ID."
```

8.10.3.6 isodate display format

`isodate` values shall be formatted to the specified accuracy.

Table 14 shall be used by the isodate display format to specify time zone conversion of the datetime.

EXAMPLE `"displayformat":"utc"`, `"displayformat":"-08"` and `"displayformat":"+10:30"`.

Table 14 — `timezone` values

`timezone` value	**Description**
"utc"	This is the default value. The DDDdata datetime shall be append a "Z".
"device"	The DDDdata datetime shall be converted to the device time zone with the time zone ±hh:mm appended.
"tag"	The DDDdata datetime shall append the `Certificate.Data.Validity.Not Before` time zone of the DDD containing DigSig certificate, if specified. Time zone conversion shall not be done.
"±hh[:mm]"	The DDDdata datetime shall be converted to the specified time zone with the time zone ±hh:mm appended.

8.10.3.7 enum display format

The enum display format is a `displaystring` using the parameters `${enumvalues}` and `${enumvaluedesc}` with the default `${enumvaluedesc}` when specified, else `${enumvalues}`.

NOTE The parameter name `"enumvalues"` is used instead of `"enumvalue"`, the singular, because `enumvalues` is the referenced field descriptor providing a single value from a list of enumerated values.

EXAMPLE

```
{"fieldid":"colour","type":"enum","description":{"en":"Vehicle colour (pantone): "},
 "enumvalues":["0018A8","F93822","00AB84"],
 "enumvaluedesc":[{"en":"blue"},{"en":"red"},{"en":"green"}],
 "displayformat":{"en":"${enumvaluedesc}(${enumvalues})"}}
```

Resulting in an

```
"displayvalue":"red(F93822)" displayed as "Vehicle colour (pantone): red(F93822)"
```

`enumvaluefields` shall be handled as an independent inserted struct as specified in 8.10.3.8.

8.10.3.8 struct display format

The struct display format has the following rules:

- The struct fields are handled independently in a hierarchical manner when neither `structjoin` nor `displayformat` is specified.
- Without `structjoin` (see 9.3): The struct display format is used to combine the `displayvalue` of the struct fields into a single a `displayvalue` using the `${<fieldid>}` parameter method to select the display value string of a struct field. The DDDdataDisplay struct shall be collapsed. Omitting and duplicating field `displayvalue` strings are allowed.
- With `structjoin` (see 9.3): The joined DDDdata value, which is a JSON string, is displayed as is.

8.10.4 DDDdataDisplay generation

```
DDDdataDisplay ⇐ GenerateDDDdataDisplay(DDD, DDDdata, <language tag>, <time zone>)
```

DDDdataDisplay shall be generated with the following steps:

a) Start with a DDD.

NOTE The structure of DDDdataDisplay is the same than the DDD which makes this processing intuitive.

b) Prune every `displaystring` according to the specified language tag, e.g.:

```
"postverify":
 {"en":"For more information contact the Department of Education.",
  "af":"Vir meer inligting kontak die Department van Opleiding."}
```

Based on the selection of the language tag "af", the value of `postverify` becomes:

```
"postverify":"Vir meer inligting kontak die Department van Opleiding."
```

c) Add the `displayvalue` pair for each leaf field in `datafields` formatted as specified by `displayformat`.

```
"displayvalue":<field value formatted for display as a JSON string>
```

NOTE 1 Each leaf within `datafields` shall receive a `displayvalue` pair.

NOTE 2 Ensure the datetime fields are presented in the specified time zone.

d) Perform `structjoin`, where specified, by creating a single string for the `struct displayvalue` as specified by the `structjoin`. Prune the fields.

e) Append/assign to `description` the non-null value of the appropriate `nulldescription` or `nonnulldescription` using space-dash-space (" – ") as the separator.

NOTE 3 The default value for `description`, `nulldescription` and `nonnulldescription` is null.

f) For a non-null `privatecontainer` (see 9.5) append/assign to `description` the non-null value of `privatecontainerdesc` using space-dash-space (" – ") as the separator.

NOTE 4 The default value for `description` and `privatecontainerdesc` is null.

g) Purge all field descriptors excluding `fieldid`, `fieldname`, `description` and `displayvalue`.

h) If a field does not contain a `fieldname` pair, then `fieldname` with the value of `fieldid` shall be added.

8.11 Structured document processing

A structured document is a "template" to be populated with DDDdata formatted for display (see 8.10). The common script parameter notation `${<fieldid>}` shall be used to place a field value in the structured document. Omitting and duplicating field display values are allowed.

An array field (see 8.9.2.2) shall be handled in one of the following ways:

- Not indexed: Concatenate the display values using a single space. The display values of each field in a `struct` field shall be concatenate first.
- Indexed: The parameter notation shall be specified with an index using the following syntax:

  `${<fieldid>.<index>}` with `index` an unsigned integer starting at 0.

  The display value replacement shall be by index.

  Indexes without values shall use an empty string (`""`).

  Indexes not included in the template shall be ignored.

A structure document is processed using the following steps:

a) Select the structured document based on language and locale from the `structureddoc` list (see 8.5).

b) Generate DDDdisplayData (see 8.10.4).

c) In the selected structured document replace all occurrences of matching `${<fieldid>}` with the display value of the matching `<fieldid>` from the DDDdisplayData. Non-matching `<fieldid>`s shall not be replaced.

NOTE The specification of `displaystring` caters for some formatting, placeholder size and alignment. It is however common to use a text based structured document which is rendered by an application. This rendering is out of scope for this document.

EXAMPLE See Annex O.

8.12 Application field specification by codebook

Application fields may be specified using a codebook containing a reverence list of application fields specifications.

NOTE 1 The use of a codebook specification reduces the risk of errors, ensures interoperability, and improved the readability of a DDD.

A codebook referenced field shall be included with the following field syntax:

```
"fieldref":[<fieldid reference>,<codebook URL>]
```

The `fieldref` shall be resolved by looking the field description (see 8.8) up in the codebook and directly replace it in the DDD.

The DDD included in a DigSig certificate shall not contain a `fieldref`.

NOTE 2 A codebook can change after the DigSig certificate was issued, which can result in a field specification conflict.

The codebook shall be a JSON construct containing the field `20248fields` with the following syntax:

```
"20248fields":<an array of application field specifications>
```

`20248fields` may be on any level in the codebook. The resolver shall use the first occurrence of `20248fields` and the matching `fieldid` value.

EXAMPLE 1 A DDD using a codebook.

```
{"digsiginfo":
  {"specificationversion":"ISO/IEC 20248:2022",
   "daid":"QC DGSG",
   "cid":1023,
   "dauri":"https://dauri1.20248.info",
   "verificationuri":["https://v1.20248.info"],
   "revocationuri":"https://dauri1.20248.info/revoked",
   "preverify":{"en":"This is a signed ISO UII"}},
 "datafields":
  [{"fieldid":"specificationversion"},
   {"fieldid":"dauri"},
   {"fieldid":"daid"},
   {"fieldid":"cid"},
   {"fieldid":"signature"},
   {"fieldid":"timestamp","range":"2022-01-01T00:00:00.0..+4]"},
   {"fieldref":["UII","https://UMcodebook.rainrfid.org"]},
   {"fieldref":["TID","https://UMcodebook.rainrfid.org"]}]}
```

EXAMPLE 2 A codebook.

```
{"20248codebookname":"RAIN RFID - DigSig in UM",
 "version":"0.0.1 example",
 "sha256":"ygkuewhdf98234yrtuhfgcin843hvlnnjcfsdzhc98437cnt839u42jnc98437p34m==",
 "description":"RAIN RFID fields are specified in this codebook. The 20248 DigSig
is stored in UserMem identified with the DSFID 0x11.  The UII-96 and TID values are
specified. The UII is obtained by the application during INVENTORY of the tags. The TID
must additionally be ACCESSed using a read command after INVENTORY.",
 "20248fields":
  [{"fieldid":"UII-96",
    "type":"bstring",
    "binaryformat":"{96}",
    "pragma":{"readmethod":{"methodname":["ISO/IEC 18000-63","INVENTORY"],
                            "methodmemory":[1],
                            "methodoffset":2,
                            "methodlength":96}}},
   {"fieldid":"TID",
    "type":"bstring","binaryformat":"{96}",
    "pragma":{"readmethod":{"methodname":["ISO/IEC 18000-63","ACCESS"],
                            "methodmemory":[2],
                            "methodoffset":0,
                            "methodlength":96}}}]}
```

9 Pragmas (field directives)

9.1 General

A pragma defines additional actions regarding the handling of the field.

Pragmas shall NOT be nested.

The pragmas of a field are specified with the `pragma` field. All pragmas are optional.

```
"pragma":
 {"entertext":null,
  "structjoin":<structjoin format>,
  "readmethod":<readmethod description>,
  "privatecontainer":<private container description>,
  "startonword":<WORD size>}
```

9.2 entertext

`entertext` instructs the DecoderVerifier application to request a field to be entered by the operator.

NOTE `entertext` is a useful method to obtain information which needs to be verified by an operator or which cannot easily be obtained automatically, e.g. a personal identification number (PIN) or even a password can be used, with `entertext` to prove ownership of an item or a ticket.

`entertext` may only be used with singular fields, with or without cardinality.

`entertext` shall NOT be used with `struct,` but may be used with the fields of the `struct`.

`entertext` may be used with a `structjoin struct` in which case the struct is treated as a `string` from which the `struct` fields values are obtained.

`entertext` shall NOT be used with `readmethod`.

The input format of a field type shall be as follows:

- `boolean`: The displayformat values shall be used in a selection list.
- `unsignedint`: A non-negative integer. Hex and base64 are not allowed to prevent type confusion.
- `number`: A decimal point number (computer format) with the characters minus, space, comma, point and "0" to "9". The number shall be converted to the appropriate `decimalunit` using `truncate`.
- `string`: The valid characters as specified with `range`.
- `bstring`: Entered as a string and converted to `HexString` (see 4.4). The input type may be auto detected or selected with a check box.
  - Binary: Valid characters are zero ("0") and one ("1") with a whitespace as a separator.
  - Hex: Valid characters are zero ("0") to nine ("9"), a case insensitive "A" to "F" with colon ":" and whitespace separators. The string may start with "0x" or ":".
  - Base64: All characters shall be one of the valid Base64 characters of the regular and URL safe alphabets (see RFC 4648). The characters may be entered with and without padding.
- `digsigenv`: Entered as a `bstring` with the default format Base64.
- `date` and `isodate`: Entered in the date and time format of the capture device and converted to UTC in the `date` or `isodate` specified format and accuracy.

— enum: The enumeration is selected from a selection list using the enumvaluedesc values, whereafter, if specified, the enumeration fields will be processed.

9.3 structjoin

structjoin is used with struct to join the structure's DDDdata field values into a single string to be used by DDDdataTagged and DDDdataDisplay in which case the structure is collapsed. DDDdata and SigData shall contain the struct values NOT joined.

NOTE The intended use of structjoin is to optimize multi-language alphanumeric encodings and include simple formatting characters, typically found in serial numbers. The scope of structjoin is limited to this intent.

structjoin uses a format string to join the DDDdata values of the structure. The common script parameter notation ${<fieldid>} shall be used to place a field value in the string. Omitting and duplicating field values are allowed.

EXAMPLE In the following field description, the struct licencenumber contains the fields 999part with the value "345" encode with 4 bits per character (the range is [0-9]) and AAApart with the value "ABC" encode with 5 bits per character (the range is [A-Z]). The licence number format is "GB 999 AAA" resulting in "GB 345 ABC" which will be encoded into 6 bits per character when not using structjoin, which is 6 × 10 = 60 bits. With structjoin the encoding will use 3 × 4 + 3 × 5 = 27 bits.

```
{"fieldid":"licencenumber",
 "type":"struct",
 "pragma":{"structjoin":"GB ${999part} ${AAApart}"},
 "fields":
  [{"fieldid":"999part",
    "type":"string",
    "binaryformat":"{3}",
    "range":"[0-9]"},
   {"fieldid":"AAApart",
    "type":"string",
    "binaryformat":"{3}",
    "range":"[A-Z]"}]}
```

9.4 readmethod

readmethod is used to specify the read method of a field. Each readmethod results in an additional DataSnip. The read method is data carrier specific. The readmethod descriptors are specified in Table 15.

Table 15 — readmethod field descriptors

Name	Purpose and value	Default	Type	Inclusion
methodname	The standard specification name for the read method (s). See 5.1 for the format. EXAMPLES ["ISO/IEC 18004"] ["ISO/IEC 18000-63", "ISO/IEC 29167-10"]	-	JSON array of JSON strings	compulsory
methoddesc	A display string to optionally instruct an operator, e.g. where a data carrier is located.	null	displaystring	optional
methodmemory	The memory segment identifier is a sequence of unsigned integers and text labels identifying the hierarchical layers of the selected data segment starting from the root parent.	[0]	JSON array of unsigned integer and string	optional
methodsnipencoding	Encoding method of the snip: BINARY or TEXT.	BINARY	JSON string	optional
methodoffset	The start of the field bit/character encoding in the memory segment.	0	unsigned integer	optional

Table 15 *(continued)*

Name	Purpose and value	Default	Type	Inclusion
methodlength	The specified number of bits/characters to read. methodlength may be omitted for data carriers which have a length field encoded as part of the read/interrogation protocol.	-	unsigned integer	optional

readmethod shall use the following syntax:

```
"readmethod":{"methodname":<method name>,
              "methoddesc":<an operator instruction>,
              "methodmemory":<memory segment location>,
              "methodsnipencoding":<"BINARY"|"TEXT">,
              "methodoffset":<an unsigned integer for the bit or character offset>,
              "methodlength":<length to be read>}
```

The null bit of the binary encoding of a nullable field using a readmethod pragma shall be contained in the parent Snip.

EXAMPLE 1 ISO/IEC 18004 (QR codes) only have one "memory" page which is read in a singular read.

```
"readmethod":{"methodname":["ISO/IEC 18004"],"methodsnipencoding":"TEXT"}
```

EXAMPLE 2 A DataSnip from ISO/IEC 18000-63 user memory:

```
"readmethod":{"methodname":["ISO/IEC 18000-63"],
              "methodmemory":[3],
              "methodoffset":128,
              "methodlength":128}
```

EXAMPLE 3 A data set encoded in ISO/IEC 15434 syntax, encoded in a QR Code, contains the data identifiers: 25S (serial number), 1P (item identification), 16D (production date) and 2E (max temperature):

[)>$^{R}{}_{S}$06$^{G}{}_{S}$**25S**QCELMI1234567XYZ$^{G}{}_{S}$**1P**12345NM$^{G}{}_{S}$**16D**20120326$^{G}{}_{S}$**2E**80$^{R}{}_{S}$^{E}O$_{T}$.

A readmethod can obtain the data of the data identifier by using the data identifier.

```
{"fieldid":"production_date",
 "fielddescription":{"en":"date of production"},
 "type":"string","binaryformat":"{8}",
 "pragma":{"readmethod":{"methodname":["ISO/IEC 18004","ISO/IEC 15434","ISO/IEC 15418"],
                         "methodsnipencoding":"TEXT",
                         "methodmemory":["16D"]}}}
```

This readmethod delivers the data (i.e. the production date): "20120326".

NOTE 1 The local application converts the bit location and length to the WORD boundaries of the data carrier.

NOTE 2 Variable length fields are complex to use with data carriers which do not provide a length value.

9.5 privatecontainer

privatecontainer is used to specify an encrypted struct. privatecontainer shall only be used with struct.

A struct with a privatecontainer shall be handled as a singular bstring and used as such in all DDDdata structures (see 8.2). The struct description shall be used by an authorized application to decode the privatecontainer struct after it was decrypted.

The scope of a decoded private container should be limited to the authorized application.

The syntax of privatecontainer shall be as follows:

```
"privatecontainer":
 {"privatecontaineruri":<the private container URI>,
"privatecontainerdesc":<an operator instruction>}
```

The `privatecontainer` descriptors are specified in Table 16.

Table 16 — `privatecontainer` field descriptors

Name	Purpose and value	Default	Type	Inclusion
privatecontaineruri	The URI where an authorized application will find the methods and keys to decrypt a private container.	-	JSON string	compulsory
privatecontainerdesc	Provides optional instruct an operator, e.g. where a data carrier is located.	null	displaystring	optional

9.6 startonword

`startonword` is used to align a field value binary encoding on the next WORD boundary.

The value of `startonword` shall be an unsigned integer providing the WORD size in bits.

NOTE This is useful for reprogrammable fields, typically with `bsign` = `false`.

EXAMPLE `"startonword":16`

Annex A
(normative)

Test methods

A.1 DigSig certificate format

A.1.1 General

This test verifies conformance of a generated DigSig certificate. This test shall not verify the validity of the cryptographic components. It shall verify conformance to the format.

A.1.2 Test configuration

The test samples shall be provided in the certificate format as specified in ISO/IEC 9594-8 and this document.

A.1.3 Test method

The test shall be performed by one of the following methods:

- By inspection: An inspector shall inspect a printed version of the sample for conformance with this document.
- By norm application: A norm application, typically a norm DigSig DecoderVerifier, shall be used to interpret the sample. A successful interpretation shall mean "PASS"; if not successful, it shall mean "FAIL".

A.1.4 Test report

The test report shall record

- the test authority,
- the date of the test,
- the method of test, and
- a list of samples, with each sample marked "PASSED" or "FAILED".

A.2 DigSig Data

A.2.1 General

This test verifies the DigSig Data as stored in a data carrier in the RAW or URI DigSig formats.

A.2.2 Test configuration

Test samples shall be provided with valid DigSig certificate(s). The test sample shall be stored in the appropriate data carrier(s).

A.2.3 Test method

The test authority shall ensure that it can read the data carriers on which the test data are stored.

The following steps shall be performed:

a) The data on the data carrier shall be read in the prescribed manner.

b) The conformance of the DigSig shall be verified by inspection or with a norm application. If the verification fails, then the test shall be stopped.

c) The DigSig Data shall be verified by inspection or with a norm application against the DigSig certificate referenced by the CID in the envelope of the test sample.

A.2.4 Test report

The report shall record the following

- the test authority,
- the date of the test,
- the method and scope of the test, and
- a list of samples, with each sample marked "PASSED" or "FAILED".

A.3 DigSig DecoderVerifier

A.3.1 General

This test verifies the conformance of a DigSig DecoderVerifier against a representative DigSig certificate.

A.3.2 Test configuration

For the DigSig DecoderVerifier to be tested, the following shall be provided by the testee:

- Test by norm application: The executable loaded on the intended platform with all the required peripherals.
- Test by inspection: The full source code with a description of all the third-party executables used.

Only third-party executables used widely in other applications shall be allowed to "PASS" as object code or executable.

A.3.3 Test method

A.3.3.1 Test by inspection

The test authority shall inspect the source code for conformance and record the full flow of data within the executable in detail.

A.3.3.2 Test by execution against a norm data set

The test authority shall prepare a sample data set that mimics the intended use of the test DigSig DecoderVerifier. This data set shall contain at least five different DigSig certificates, each supplied with at least 100 DigSig samples programmed in the intended data carrier. Random tampering shall be applied to at least 10 % of the samples. The test shall indicate the rejections.

A.3.4 Test report

The report shall record the following

- the test authority,

- the date of the test,
- the method of test, which includes the test data set, and
- a description of the sample (including the platform and peripherals used in the test) as tested, marked either "PASSED" or "FAILED".

A.4 DigSig EncoderGenerator

A.4.1 General

This test verifies the conformance of a DigSig EncoderGenerator against a set of valid DigSig Data Descriptions and DigSig certificates.

A.4.2 Test configuration

For the DigSig EncoderGenerator to be tested, the following shall be provided:

- For test by inspection: the full source code shall be available with a description of all third-party executables used. Only third-party executables that are used widely in other applications shall be allowed to "PASS" as object code or executable.
- For test by execution: the DigSig EncoderGenerator shall be available with its executable loaded on the intended platform with all the required peripherals.

A.4.3 Test method

A.4.3.1 Test by inspection

The test authority shall inspect the source code for conformance and record in detail the full flow of data within the executable.

A.4.3.2 Test by execution against a norm data set

The test authority shall prepare a sample data set that mimics the intended use of the test DigSig EncoderGenerator. This data set shall contain at least five different representative DigSig Data Descriptions, each supplied with at least 10 DigSig data samples.

The following test steps shall be performed for each of the DigSig data samples and the result inspected for conformance:

a) Generate the DigSig certificate.

b) Generate the test RAW and URI DigSigs.

c) Inspect the DigSigs for correctness by inspection or by norm DigSig DecoderVerifier in accordance with A.2.

A.4.3.3 Test report

The report shall record the following

- the test authority,
- the date of the test,
- the method of test, which includes the test data set, and
- a description of the sample (including the platform and peripherals that were used in the test) tested, marked either "PASSED" or "FAILED".

Annex B
(informative)

Example DigSigs

B.1 General

The following examples illustrate the many uses of this document. It is not meant to be an exhaustive set but rather an attempt to illustrate the flexibility and control of the specification. The following examples are simplified and provided in the DigSig shorthand (B.2). The examples are neither representative nor prescriptive in any way.

It should be noted that when a DigSig verification is accepted, the following is valid:

— The data read from the data carrier or received from a device have been decoded according to the directive of the authority of the data.

— The data read are unaltered from when they were written.

— The data were authorized (signed) at the instance of the timestamp.

B.2 DigSig shorthand syntax

A DigSig described in the specified DDD format (see Clause 8) is often too elaborate for easy reading. For shorthand purposes, the following syntax may be used:

```
[name of DigSig]DigSig[data carrier type]{daid, cid, fields}
```

```
fields |= the set of fields
```

NOTE 1 `daid`, `cid`, `signature` and `timestamp` are compulsory fields.

A DigSig field which is not signed is depicted by strikethrough text, i.e. ~~NotSignedField~~. The compulsory field `signature` is an example of a field, in the DigSig, which is not signed.

Since `daid`, `cid`, `signature` and `timestamp` are compulsory fields of the DigSig, it need not be shown in the shorthand. The `daid` and `cid` shall be the first fields. The `signature` and `timestamp`, by default, follow the `daid` and `cid` in that order. The `signature` and `timestamp` fields may be placed anywhere in the DigSig, but then it shall be shown.

EXAMPLE 1 `ItemDigSigNFC{daid, cid, signature, timestamp, ItemID, OwnerID}`

`daid` and `cid` shall be the first 2 fields. It can therefore be safely omitted in the shorthand. The default position for `signature` and `timestamp` are third and fourth in which case it can also be omitted in the shorthand.

EXAMPLE 2 `ItemDigSigNFC{ItemID, OwnerID}`

NOTE 2 Examples 1 and 2 represent the same DigSig.

`signature` and `timestamp` are explicitly placed to change it position in the DigSig. This is useful with multi-page memory AIDC data carriers.

EXAMPLE 3 `ItemDigSigNFC{ItemID, OwnerID, signature, timestamp}`

NOTE 3 In Example 3 the **signature** and **timestamp** are explicitly placed at the end of the DigSig.

{field…} groups a set of fields. This group can be prefixed with another memory location or data carrier which indicates it is not part of the CIDSnip and is read using a read method, i.e. IDcardLinearBarcode{NotInCIDSnip}. In the next example `signature` and `timestamp` are placed into MB11 of an ISO/IEC 18000-63 tag.

EXAMPLE 4 `ItemDigSigRAIN{ItemID, OwnerID, MB11{signature, timestamp}}`

NOTE 4 In Example 4 the term RAIN is an accepted industry term for ISO/IEC 18000-63.

NOTE 5 In Example 4 `signature` and `timestamp` are explicitly placed in MB11 of the ISO/IEC 18000-63 tag.

EXAMPLE 5 `ItemDigSigNFC{ItemID, OwnerID, entertext{SecretPIN}}`

NOTE 6 In Example 5 the SecretPIN is secret since it is not part of any Snip. It is to be entered by the operator, typically from memory.

The square brackets "[]" indicate an array of values for a field, i.e. "field1" indicates field1 has one value. "field1[]" indicates field1 has none, one or more values.

The symbols "…" and ",…" may be used to indicate none and more, and one and more records for a list.

EXAMPLE 6 `ContainerItemList{ContainerID, List{{ItemID, ItemDesc}…}[]}`

To aid readability, fields can be grouped under a group name and expanded on a separate line. Similarly, it is recommended that fields of DataSnips be described in this way.

EXAMPLE 7 `ContainerItemList{ContainerID, List{}}` with `List{{ItemID, ItemDesc}…}` stored in the QR code on the waybill.

B.3 Use examples

B.3.1 A simple timestamp

A simple timestamp can be created with an empty DigSig:

```
Timestamp{}
```

The DDD for this simple timestamp is as follows (note the use of defaults to make it compact):

```
{"digsiginfo":
  {"specificationversion":"ISO/IEC 20248:2022",
   "dauri":"https://thetimestamp.org",
   "daid":"A TIME",
   "cid":"123",
   "verificationuri":["https://verify.thetimestamp.org"],
   "revocationuri":["https://revoke.thetimestamp.org"],
   "acceptverify":{"en":"This is a valid timestamp."},
   "rejectverify":{"en":"The timestamp cannot be trusted."}},
 "datafields":
  [{"fieldid":"specificationversion"},
   {"fieldid":"dauri"},
   {"fieldid":"daid"},
   {"fieldid":"cid"},
   {"fieldid":"signature"},
   {"fieldid":"timestamp","range":"[2022-01-01T00:00:00..2024]"}]}
```

The decoding and verification of this DigSig will verify the Domain Authority and the time at which the timestamp was generated.

A device under the control of a Domain Authority may like to timestamp data it created and prove that the data originated from it. The DigSig will then look like this:

```
TimeStamp{DeviceID, DeviceGeneratedInfo{}}
```

B.3.2 An authentic digital-twin tag

A digital-twin label/tag is typically an AIDC data carrier which only contains a reference to a cloud service where all the relevant information of the tagged object resides. It is important that such a reference is unique and that the reference contains an indicator of which "cloud" to use.

A "timestamp" as discussed in B.3.1 is in fact unique as long as only one signature is generated within the accuracy of the timestamp, e.g. one a second in the default setting. The dauri provides for the verifiable reference to the authoritative "cloud".

The time accuracy limitation is mitigated where the AIDC data carrier contains a trusted unique ID, e.g. the TID for an ISO/IEC 18000-63 tag, by simply including the trusted unique DI in the DigSig.

B.3.3 Barcode DigSigs

Barcodes are typically encoded with human readable text. It became common practice, in non-industrial applications, to prefix the barcode with an URI which informs the reader what to do with it. Associated data are often encoded in text or contained in a sparse proprietary data structure. The result is a large barcode, which may reduce the read quality of the barcode or may be too large for the placement.

To be noted, it is easy to tamper with a barcode by simply changing the read barcode and recreating it. This may lead to a barcode which points to a rogue URI pointing a malicious website.

A DigSigs allows for the detection barcode tampering. A DigSig however, does not provide a method to detect copying of a barcode, but it is possible to link a barcode uniquely to an object. Consider the following example:

```
ComputerOwnershipQR{ComputerSerialNumber,
                    IDcardLinearBarcode{OwnerID},
                    OwnerMemory{PIN}}
```

NOTE In the above example, the name of the field also indicates read method; IDcardLinearBarcode, in this example, is a linear barcode in the South African identification book and OwnerMemory refers to the memory of a real person. Where IDcardLinearBarcode{OwnerID} is a `struct` with a “ISO/IEC 15417” read method and OwnerMemory{PIN} is a struct with an entertext read method.

In the above example, when the barcode is read with a mobile phone, the DDD will

a) instructs the standard DigSig DecoderVerifier app on the phone to display the Domain Authority and CID for inspection,

b) instructs the standard DigSig DecoderVerifier app on the phone to display the ComputerSerialNumber for inspection,

c) requests the operator of the phone to scan the linear barcode of the ID book/card of the person claiming the ownership of computer, and

d) requests the person claiming the computer to enter a PIN.

Copying this barcode is futile since it is linked to the computer by its serial number, the ID of the owner and something only the owner knows, the PIN. Ownership of the computer can therefore be proven, that is, if the Domain Authority can be trusted.

If in doubt, the phone user may check the revocation status of the DigSig and/or perform an online verification to confirm the offline verification at the trust service of the Domain Authority of the DigSig.

B.3.4 A RFID example

Tag data copying can be detected when an electronic data carrier with a trusted unique TagID is used, e.g. an English language competency certificate example may be constructed as follows:

```
IELTS{{ACADEMIC | STANDARD}, Date, Centre,
      Candidate{Number, ID, FamilyName, Names[], NationalityENUM, FirstLanguageENUM}
```

```
    Results{Listening, Reading, Writing, Speaking, BandScore, CEFR},
    NullableTID{TagID}(null: "Note, this is a copy.", nonnull: "This is the original.")}
```

This DDD has two applications, which can either be used independently or together to facilitate the best of both approaches.

- Barcode DigSig – TagID is be set to null since a barcode does not have a TagID. This DigSig barcode is applied on the certificate. The null description: The null description "This is a copy." will be displayed with the result of the verification.
- RFID DigSig – the unique RFID TagID is included in the DigSig. The RFID tag is applied to the original certificate. The nonnull description "This is the original." will be displayed with the result of the verification.

NOTE Typically the TagID of the data carrier is stored in a different memory segment from the memory segment which is read during the first data carrier access. The DDD specifies a read method for the TagID which will result in an additional DataSnip requiring an additional data carrier read access.

B.3.5 A simple file system

A simple DigSig record system is specified for an ISO/IEC 18000-63 data carrier in the following example.

```
PrimaryDigSig{UII{CID, PrimaryFields{},
              TID{TagID},
              UserMem{Signature, TimeStamp, DigSigRecordList{}, [DigSigRecord{}…]}

DigSigRecord |= Blocksize, DigSigContainer{Start-OffsetBlock, Length-Blocks}[]

BlockSize |= Enumeration{1-bit | 16-bits | 32-bits | 128-bits} encoded in 2 bits.
```

This DDD allows the writing of many DigSigRecords to the data carrier. The following applies:

- Each record is a DigSig and therefore can independently be verified. The records need not be the same since its CID referenced DDD describes its fields and encoding.
- DigSigRecordList is not signed and can therefore be updated as and when a new record is added.
- DigSigRecordList should start on a memory WORD boundary to assist in updating its values.

B.3.6 A sequence of DigSigs

Sequences of DigSigs can be used to establish a trail of trust when accountability/ownership of an object is transferred. This trail of trust may be stored on the data carrier in a manner discussed in B.3.5. It should be noted that, in this offline example, each role player is deployed with a DigSig EncoderGenerator which falls under the public key infrastructure of a domain authority to allow them to generate DigSigs.

Let us consider the lifecycle of a vehicle licence plate, which is like the lifecycles of high value and health risk goods, e.g. scheduled medicine. To ensure that the plate conforms to all the physical and registration rules, the following steps are taken, assuming an electronic data carrier is embedded in the plate:

- The manufacturer of the blank plate (Blanker) guarantees the plates are manufactured in conformance with relevant blank plate specifications. These plates are sent to Embossers.
- An Embosser will add the licence number to the plate after the Embosser verified that the blank is genuine, and the licence number is legal. These plates are sent to the issuing point.
- The issuing point issues the plate and mark it as issued ("live"). This may also involve that the vehicle visual parameters are added to the data carrier with an additional DigSig.

It should be noted that, in each step, the authenticity of the object is verified before the next step takes place. Also, the entity which acts on the plate should be held accountable for that action. DigSigs provide that function. A DDD for ISO/IEC 18000-63 (type C) might look as follows:

```
LicencePlateDigSig{UII{LicenceNumber^Nullable, VehicleVisuals{}^Nullable, LicenceExpiry},
                   TID{TagID},
                   UserMem{~~Signature~~, TimeStamp, BlankSerial,
                           ~~BlankDigSig{}~~^Nullable, EmbossDigSig{}^Nullable}}
```

NOTE 1 The position of the fields optimises both the reading and writing of the data carrier in the various use cases.

The steps are as follows:

a) The Blanker generates (signs) the DigSig with the following application field: BlankSerial.

b) The Embosser reads and verifies the [Blank] DigSig. If accepted, it knows it is a legal blank, manufactured by the specific Blanker who signed it, ready to be embossed. The plate is embossed with the legal licence number. The Embosser copies the [active Blank] DigSig to the BlankDigSig location and locks that part of the memory. The Embosser adds the LicenceNumber field and generates a new [active] DigSig, then writes it to the DataCarrier.

c) The Issuer reads and verifies the [Emboss] DigSig. If accepted, it knows this is a valid plate, prepared by the specific Embosser who signed it, ready to be issued. The BlankDigSig may also be verified before continuing. The Issuer copies the [active Emboss] DigSig to the EmbossDigSig location. The Issuer adds the VehicleVisuals{} and LicenceExpiry fields and generates a new [active] DigSig, then writes it to the DataCarrier. The Issuer locks the data carrier memory, as per applicable standard. The plate is handed to the vehicle owner as fully issued and active.

NOTE 2 The BlankDigSig, and likewise EmbossDigSig, remains independently verifiable, but since they are marked as bsign = false they are not required to be read for verification of the current [active] DigSig.

This can seem like an elaborate process, but it works well even in places with connectivity challenges. A plate of which the DigSig does not verify is illegal and should be confiscated and/or destroyed. Inspectors can check for illegal plates and activities anytime, at any place. If authorized and carrying the appropriate mobile equipment, inspectors can verify, or update recorded data during a physical inspection and cross-refer to a licence disc compliant with this standard to update the LicenceExpiry field. Such a data field correction requires the generation of a new DigSig. In contrast, verification may only be required to report back to the inspector's back office at a later time. Pilfered plates will also be detected when used illegally or inserted back into the process. All the role players can be held accountable because they can be identified through the DigSig they generated.

B.3.7 Selective struct

See 8.9.3.8 `enumvaluefields`.

B.3.8 Multiple DigSigs using the same base data

It is possible for different DigSigs to use the same part of data carrier. For example, a waybill barcode DigSig for a container includes the TagID of the container seal, which is read using the readmethod pragma. The container seal contains its own DigSigRFID. The DigSig will be rejected in both cases if the seal is broken.

```
WayBill^QR{ContainerInfo{}, Source{}, Destination{}, SealTID{TagID}}
ContainerSeal^63{UII{SealInfo{}}, UserMem{~~Signature~~, Timestamp}, SealTID{TagID}}
```

B.4 Signature method examples

B.4.1 General

The use of standard digital signature methods is recommended; however, standard methods are designed for large data blocks where data size is not a critical limitation. AIDC data carriers are limited in the number of bits it can store especially when considering the read scenario. This clause provides example digital signature methods, standard, standard adapted and proprietary. These examples are not provided as a recommendation for general use, the use case risk needs to be evaluated to decide if such or which example method may be applicable (see Annex G). Also, to be noted is the ever-changing improvement of computer power, which reduce the security level of a crypto method in over time. The use of X.509 by this document allows for the seamless upgrade to stronger crypto methods whenever the need arises.

A digital signature method has the following components:

- An asymmetric cypher utilizing a secret private key (or set of information) with which the DigSig is signed, and a public key (or set of information) which is almost impossible to derive from the private key with which the signature is verified.
- A hashing algorithm.

A digital signature method is typically named using these two components, e.g. ECDSAwithSHA256 or ecdsa-with-SHA256 which is identified with the OID 1.2.840.10045.4.3.2. The first method of naming is used by this document.

B.4.2 Standard digital signature methods

Table B.1 provides standard digital signature methods which may be suitable for use with DigSigs.

Table B.1 — Example standard digital signature methods

Method name	OID	Reference specifications	Signature length
ecdsa-with-SHA256	1.2.840.10045.4.3.2	IETF RFC 5480 and IETF RFC 5758	512 bits
id-ecdsa-with-sha3-256	2.16.840.1.101.3.4.3.10	NIST CSOR	512 bits
sha256	2.16.840.1.101.3.4.2.1	NIST PUB FIPS 180-3	256 bits

B.4.3 A proprietary digital signature method (ECBNwithSHA256)

Method OID: 1.3.6.1.4.1.49897.1

This method is in common use by DigSig implementations.

Description: ECBN with SHA256 using point reduction.

Information: The ECBN MIRACL C library (pre-2018) is used for the implementation.

Public key description: `(Pub(Point1(x,y),Point2(x,y)),Sys_param(Point1(x,y),Point2(x,y)))` with x and y 32-byte unsigned integers.

B.4.4 DigSig null signature — data definition verification (NULLwithNULL)

Method OID: 1.0.20248.1.1

NOTE Null encryption is a standardised practice as illustrated by ITU H.235.6.

A requirement exists for the distribution of a verifiable AIDC data description to be used by independent systems. A DigSig DDD is a schema data description distributed in a digital certificate, which makes it independently verifiable, like code-signing. However, the verification of the data is often not important, and the bit size of the signature causes read performance issues. A null signature (no signature) may be used to reduce the signature to zero bits and avoid the size issue.

NULLwithNULL has a zero-bit signature with no data verification, while the DigSig certificate provides authenticity for the data description.

The timestamp may also be null'ed (see 8.9.3.7.3).

B.4.5 Compact data error detection (NULLwithCRC32)

Method OID: 1.0.20248.1.2

Simple data error detection (not data authenticity) may be provided using CRC32 as the hash algorithm in addition to the data structure authenticity (see B.4.4). There is no public key, and the signature length is 32 bits.

NULLwithCRC32 uses the 32-bit CRC as specified by IEEE 802.3-2012. It uses the following polynomial:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The advantages are:

- Short "signature" of 32 bits.
- No security is claimed and therefore no false sense of security.
- No digital signature verification process is needed, resulting in a faster execution time.
- Data errors are detected.
- Data structure tamper/errors are detectable.
- The data structure is distributed using a managed and verifiable method, the X.509 DigSig certificate.

B.4.6 Compact elliptic curve (FP256BNwithSHA256)

Method OID: 1.0.20248.1.3

A useful 256-bit signature is achievable through the point reduction of ECBN. It provides a 100-bit security level (as of 2020). It is a well-used algorithm especially for short messages in IoT as is the case with AIDC data.

Public key description: `(Pub(Point1(x,y),Point2(x,y)),Sys_param(Point1(x,y),Point2(x,y)))` with x and y 32-byte unsigned integers.

The following example FP256BNwithSHA256 implementation steps was created using the MIRACL Core: Apache License, Version 2.0 library dated 2020.

- MIRACL Core library using FP256BN with SHA256.
- Uses the BLS signature from emerging standard https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-02.
- Cipher suite ID: BLS_SIG_FP256BNG1_XMD:SHA-256_SVDW_RO_NUL_.

An alternative implementation may use https://github.com/cloudflare/bn256. Care should be taken in the parameter preparation, hashing and hash-to-curve steps to ensure interoperability.

NOTE This example uses the data from Annex I.

a) Key generation

Generated private key:

```
0xA1356E9DC2166EA15C7755C4C282AE81DE9439880CE02541CE0C7AEF300760E9
```

Generated public key:

```
0x026C451A6645729B679916E15FA1FFA79C3D1F516A3862C9E3618530249CB6BF42527BF3770AD0D6E435
2EAC84A408BC96F93E229448766B4DA5EA2FD4E11D4374
```

b) Signing process

Extracted SigData:

```
["ISO/IEC 20248:2021","https://www.dept-edu.com","QC DGSG",110,"2024-
11-04T08:00:00","John Doe","612209498902",["2021","2022","2023"],"Bach
elors in Administration","Business School","2024-03-04",[["Structures
101","degree","B"],["Accounting 112","degree","A"],["Statistics 159","extra","A"]]]
```

Generate the signature (33 bytes):

```
0x03FD88E0782374DA3BD55BB7A847FB019C0C8B3B8D3775B7441F5DF9F9B6256A5A
```

Drop the first byte to produce a compressed signature (32 bytes to fit common RFID memory boundaries):

```
0xFD88E0782374DA3BD55BB7A847FB019C0C8B3B8D3775B7441F5DF9F9B6256A5A
```

Encoded the DDDdata after adding the signature:

```
0xC098099640006EE100FD88E0782374DA3BD55BB7A847FB019C0C8B3B8D3775B7441F5DF9F9B6256A5A0F
2B773A129BDA1B88111BD96C3631323230393439383930328C6C3684C2C6D0CAD8DEE4E640D2DC4082C8DA
D2DCD2E6E8E4C2E8D2DEDD7A13AB9B4B732B9B99029B1B437B7B65AEEEEEE5CC2C4C65CC2C65CF4C2CC91D
CA6E8E4EAC6E8EAE4CAE6406260623720B1B1B7BAB73A34B7339018989905CA6E8C2E8D2E6E8D2C6E64062
6A7300
```

Encoded the URI envelope:

```
https://www.dept-edu.com?wJgJlkAAbuEA_YjgeCN02jvVW7eoR_sBnAyLO403dbdEH135-bYlaloPK3c6E
pvaG4gRG9lsNjEyMjA5NDk4OTAyjGw2hMLG0MrY3uTmQNLcQILI2tLc0ubo5MLo0t7dehOrm0tzK5uZApsbQ3t
7Za7u7uXMLExlzCxlz0wsyR3Kbo5OrG6OrkyuZAYmBiNyCxsbe6tzo0tzOQGJiZBcpujC6NLm6NLG5kBianMA
```

c) Verification process

Decode the DigSig envelope and extracted the SigData:

```
["ISO/IEC 20248:2021","https://www.dept-edu.com","QC DGSG",110,"2024-
11-04T08:00:00","John Doe","612209498902",["2021","2022","2023"],"Bach
elors in Administration","Business School","2024-03-04",[["Structures
101","degree","B"],["Accounting 112","degree","A"],["Statistics 159","extra","A"]]]
```

Extracted the signature: (32 bytes):

```
0xFD88E0782374DA3BD55BB7A847FB019C0C8B3B8D3775B7441F5DF9F9B6256A5A
```

Verification is performed using the public key from the DigSig certificate, the extracted SigData, and alternatively the extracted signature prefixed with 0x02 or 0x03. Verification is accepted when either of the prefixed signatures verify.

Annex C
(informative)

DigSig use in IoT

The Internet of Things or IoT is an abstract of the relations and interactions between real world things (humans and objects) and the digital world. The digital world consists more and more of autonomous and intelligent devices performing tasks independent of central services. Typically, sets of these devices are under the control and management of independent domain authorities. The domain authorities increasingly direct devices through policies, relying on the device intelligence and local information, to act. These domain authorities will assume accountability of the actions of devices under their control. For this to work, devices need to trust identity, data, information, and methods. This relies on interoperability of identity, data, and methods. It should be noted that the world of IoT is a fast-changing world. Domain authorities therefore have a key requirement for agility in defining data objects and data methods while maintaining the integrity and identity of such definitions and methods. At the same time, Big Data Stores have a requirement for authentic and interoperable data to provide correct and meaningful interactions with the physical world.

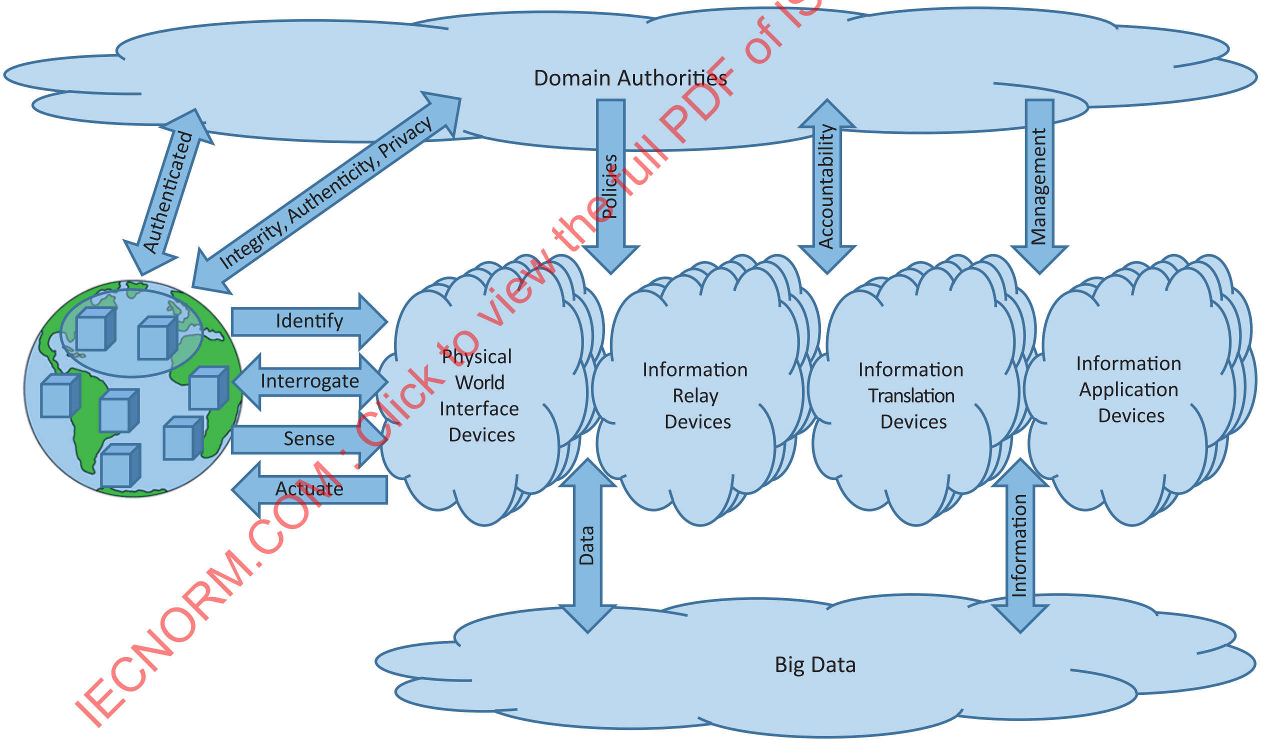

Domain Authorities
Authenticated
Integrity, Authenticity, Privacy
Policies
Accountability
Management
Identify
Interrogate
Sense
Actuate
Physical World Interface Devices
Information Relay Devices
Information Translation Devices
Information Application Devices
Data
Information
Big Data

Figure C.1 — Simplified view of the Internet of Things

Automated identification data carriers fulfil an important role in representing physical world objects in the digital world. It provides identity and information (both static and interactive) of the object and the environment of the object when connected to a sensor. For example, an aeroplane part is identified by the owner of the part. The part carries information of installation and every inspection of the part. To illustrate the requirement for interoperability of data and methods, consider that there are many independent aircraft owners and manufactures. There are also many independent part inspectors and service entities. Each of them has a requirement to access the data on the data carrier, to trust the data on the data carrier and be trusted to update the data carrier correctly. The world of commercial aircraft is successfully achieving such goals, but a single domain authority is working hard to reach these goals. IoT expands these requirements to domains where such rigidity is not achievable.

In the context of this document, the role players in Figure C.1 perform the following functions (see Figure 1 for context):

— Physical world objects are tagged with a data carrier.

— A domain authority, which is accountable for the object, programs the data carrier with a DigSig containing identity data and optional information about the object. The domain authority has previously published a DigSig certificate which describes the data structure and read methods of the DigSig.

EXAMPLE 1 `DigSig{ObjectID, [ObjectInfo], [PrivateInfo]}`

NOTE 1 The DigSig certificate is a verifiable and trusted directive from an independent domain authority, instructing an autonomous device how to access data carriers on objects from that domain authority.

NOTE 2 The DigSig inherently identifies the domain authority of the object.

— The data carrier is accessed and read by a Real-World Interface Device, which can be under the control of a different domain authority than the domain authority of the data carrier. In this case, because of its access to the DigSig certificate, the Device can decode and verify the data carrier data. It can also read the encrypted data if it has been assigned authenticated access to the DigSig private container.

NOTE 3 If so configured, the data carrier remains untraceable, and the private containers remain private until the Device has been authenticated using the method specified in the DigSig certificate.

NOTE 4 The Real-World Interface Device can also, by its own logic or under the directive of another device, write information to the data carrier.

In IoT, it is possible that other intelligent devices need the information of a data carrier access event to fulfil their functions. These devices need to trust the access event information, especially since the data carrier is not present. This is achieved by:

— Every Real-World Interface Device is issued with a X.509 identity by its domain authority. The associated DigSig certificate describes the access events of the device. Likewise, every device that produces data should be issued with a X.509 identity by its domain authority. The associated DigSig certificate describes the information produced by the device.

— The Real-World Interface Device digitally signs the access event. It then makes the event available to other IoT devices, which can then verify the data accessed, when, by whom and if it was, how it was manipulated.

EXAMPLE 2 In the example the AccessEventInfo may be in a private container for security purposes.

```
DigSig{AccessEventTimestamp,[AccessEventInfo],
        DigSig{ObjectID,ObjectInfo,PrivateInfo}}

AccessEventInfo |= {[AccessEventInfoStatic],
                    [AccessEventInfoActions],
                    [AccessEventSensorInfo]}
```

NOTE 5 The DigSig inherently identifies the domain authority of the Device and therefore also the DigSig certificate whereby the read event DigSig can be interpreted.

— The relaying devices can verify the DigSig(s) and sign it as being relayed by a specific device at a specific time in the manner described above. However, it will not be able to access private containers without proper authentication.

— Information translation devices typically join and translate access events with other information to make the access event more useful. The result of joining, translation and aggregation also should be signed.

— Following this chain, the Application device can authenticate the data carrier data without the need of directly interrogating the data carrier. It can also verify all additional information and changes to the information, in relation to the access event, before it uses the information.

This chain of DigSigs provides accountability of actions across all devices, even if these devices are under the control of independent domain authorities. Interoperability of data structures and read methods are maintained. Data structure and method changes are handled by issuing a new DigSig certificate with the new structures and methods (see Figure 3).

Annex D
(informative)

Typical DigSig EncoderGenerator device architecture

D.1 General

This annex should be read with Annex E.

D.2 Architecture

Figure D.1 illustrates a typical DigSig EncoderGenerator device architecture. The DigSig encoding and generation services typically consist of an Issuer device and the EncoderGenerator device. This separation ensures proper segmentation of functions and allows for efficient information security measures. The private key used for the generation of the signature should be kept secured within the EncoderGenerator device.

The Issuer device is typically responsible for

- the assembly of the DigSig data to be signed,
- the request for a DigSig, and
- the programming of the data carriers.

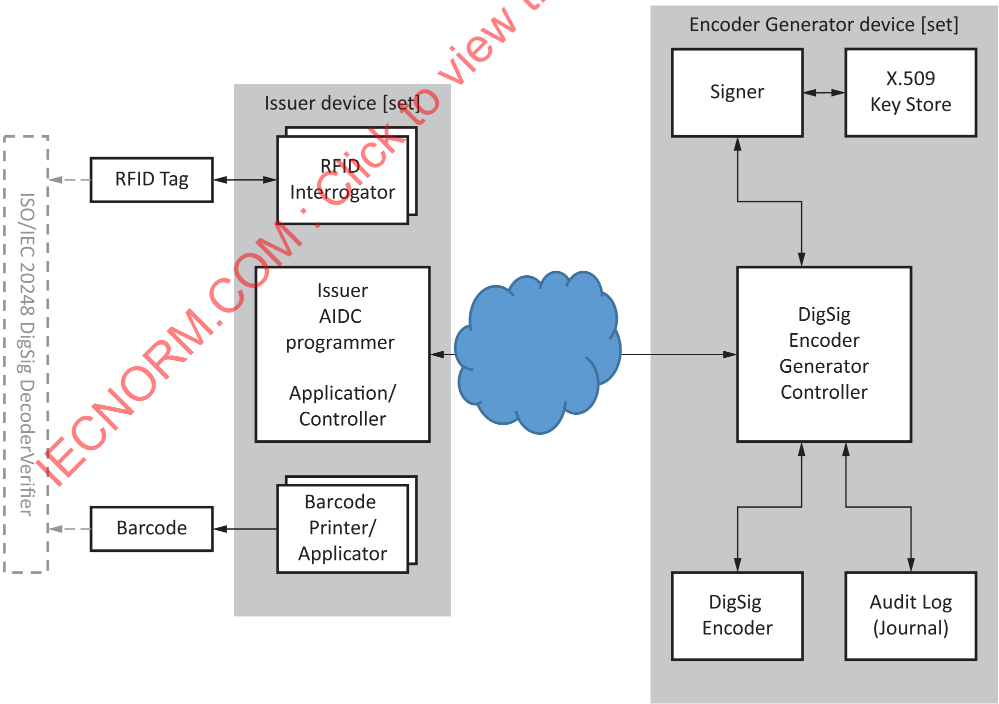


Figure D.1 — Typical DigSig EncoderGenerator use architecture

The EncoderGenerator device is typically responsible for

- the protection of the private key,

- the generation of the X.509 key pair,
- journaling of all DigSigs generated,
- the generation of the Timestamp, SigData and Signature, and
- the encoding of the DigSig.

A DigSig issuing service, incorporating one or more EncoderGenerator devices is typically responsible for

- controlling and managing the DigSig EncoderGenerator device(s),
- the validation of the DigSig request and the credentials of the requestor (and issuing device),
- the process of the DigSig generation,
- the checking of the DigSig before release to the Issuer device,
- creating an audit trail of DigSig requests, DigSigs generated and error conditions,
- acts as a security shield for the EncoderGenerator device,
- optionally provides the online verification services, and
- optionally provides the online DigSig revocation services.

This distributed architecture may be simplified if all the functions are local to a specific point, e.g. on a factory line or in a distribution centre.

It is recommended, to ensure signature fidelity, that the DigSig be verified for valid verification before it is programmed on the data carriers. This means the DDDdata should undergo the full cycle of encoding and decoding before being signed and released. SigData should be generated from the decoded DigSig.

D.3 Example interface

D.3.1 Principles

This example DigSig EncoderGenerator interface is widely used to provide a TCP/IP command-response (message) service. It is used to encode and sign data and generate the DigSig as specified by a DigSig Data Description (see 8.5).

All commands shall be responded to and processed in sequence.

DigSig EncoderGenerator is typically supported by the following utilities:

- DigSig Data Description checker.
- DigSig certificate sign-request generator.
- DigSig certificate signer, typically a function of an X.509 Certificate Authority.

D.3.2 Version

The interface notifies the application which ISO/IEC 20248 specifications and which interface implementations are supported by the service, using `Version` in the information notice.

The versions for the interface specified in this document are:

- Specification: "ISO/IEC 20248:2022".
- Interface: 2.

The version field has the following format:

```
"Version":{"Specifications":[list of supported ISO specifications],
           "InterfaceSupport":[list of supported interface versions],
           "Implementation":
            {"VendorName":"<vendor name>",
             "Version":"<major>.<minor>.<patch>",
             "Build":"<build number"}}
```

EXAMPLE

```
"Version":{"Specifications":["ISO/IEC 20248:2018","ISO/IEC 20248:2022"],
           "InterfaceSupport":[1,2],
           "Implementation":
            {"VendorName":"20248 Technologies",
             "Version":"2.2.3",
             "Build":"231921-alpha"}}
```

D.3.3 Interface message

D.3.3.1 Method

The interface uses a command-response method of which the initial "Info" message is a response to a successful connection.

D.3.3.2 Message framing

The interface message is a single inline JSON object containing a set of fields in the following format:

```
{"<message name>":<message data>}EOL with EOL |= <LF> | <CR> | <LF><CR> | <CR><LF>
```

NOTE Field order within <message data>, as with all JSON objects, has no meaning.

D.3.3.3 Binary values

See 4.4.

D.3.3.4 Communication start-up

On receiving a connection, the service shall send an information notice.

D.3.3.5 Communication termination

Empty messages (an EOL preceded with no text) shall terminate the connection with no error.

D.3.3.6 Message error

Commands which are not valid JSON objects shall be discarded with the following response:

```
{"Error":{"Description":"Incomplete command"}}
```

D.3.4 Certificate store

The certificate store contains the desired DigSig certificates.

The certificate store may contain the associated private keys.

The private keys should be kept secret to ensure authenticity of the system. Only one copy of a private key should ever exist. If the private keys are stored in the certificate store, then the certificate store should also be kept secret.

An independent secure and trusted service should manage this store.

The EncoderGenerator shall watch the store for a change on which it shall enact the change and notify connected applications of the change.

D.3.5 Keys and DigSig certificates

D.3.5.1 General

The generation of new keys and DigSig certificates, using this interface, shall use the following steps, see Figure 2:

a) The application requests the generation of a new key pair. The EncoderGenerator generates the specified key pair and

- — stores the key pair securely, and
- — return the public key.

b) The application adds the correct DAID and CID to the DDD.

c) The application generates a DigSig certification request by including the public key and the DDD.

d) The application submits the DigSig certification request to an appropriate X.509 Certification Authority.

e) The resultant DigSig certificate is

- — published appropriately, and
- — provided to the EncoderGenerator upon which it matches the private key by using the stored public key and the public key in the certificate.

f) The EncoderGenerator reports, using the info message, its readiness to perform signing for that DigSig certificate.

NOTE A key request is not required when any of the NULL encryption methods are used.

D.3.5.2 Key request

The following command shall be used to generate a new key pair, securely store the private key and provide the public key to be associated with a DDD:

```
{"KeyRequest":<crypto signature method>}
```

The response has the following format:

```
{"KeyRequest":{"ResponseCode":{"Code":<JSON number - code as per Table D.1>,
                               "Desc":<JSON string - error as per Table D.1>},
               "PubKey":<Public key in base64>}}
```

Table D.1 — Key request error codes

Error	Description
0	Success
1	Failed: Method not supported
2	Failed: Key generation error
3	Failed: Key storage error

NOTE The method knows the key length. It is therefore redundant information and excluded from the interface.

D.3.5.3 Add a DigSig certificate

The following command shall be used to add a DigSig certificate:

```
{"AddCert":<the DigSig certificate in the PEM format>}
```

The response has the following format:

```
{"AddCert":{"ResponseCode":{"Code":<JSON number – code as per Table D.2>,
                            "Desc":<JSON string – error as per Table D.2>}}}
```

Table D.2 — Add a DigSig certificate error codes

Error	Description
0	Success
11	Failed: No private key match
12	Failed: Certificate storage error
13	Failed: DDD decode error

The application should perform an information request to obtain the DataSnip configuration.

D.3.5.4 Delete a DigSig certificate

The following command shall be used to delete a DigSig certificate:

```
{"DelCert":{"DAID":<DAID>,"CID":<CID>}}
```

The response has the following format:

```
{"DelCert":{"ResponseCode":{"Code":<JSON number – code as per Table D.3>,
                            "Desc":<JSON string – error as per Table D.3>}}}
```

Table D.3 — Delete a DigSig certificate error codes

Error	Description
0	Success
21	Failed: Cert not found
22	Failed: Deletion error

D.3.5.5 List the key pairs

The following command shall be used to list the public key of the generated key pairs with the linked DigSig certificates identified by the DAID and CID:

```
{"ListKeys":null}
```

The response has the following format:

```
{"ListKeys":{"ResponseCode":{"Code":<JSON number – code as per Table D.4>,
                             "Desc":<JSON string – error as per Table D.4>},
             "PubKeys":[{"PubKey":<Public key in base64>,
                         "Certs":[{"DAID":<DAID>,"CID":<CID>}…]},…]}}
```

With "No keys", the field `PubKeys` may be omitted. If included in the response, then it shall have an empty array as a value, `"PubKeys":[]`.

No linked certificates shall be indicated with an empty array, `"Certs":[]`.

Table D.4 — List key pairs error codes

Error	Description
0	Success
31	Failed: No keys
32	Failed: List error

D.3.5.6 Delete a key pair

The following command shall be used to delete a key pair and its linked DigSig certificates:

```
{"DelKey":<Public key in base64>}
```

The response has the following format:

```
{"DelKey":{"ResponseCode":{"Code":<JSON number - code as per Table D.5>,
                           "Desc":<JSON string - error as per Table D.5>}}}
```

Table D.5 — Delete a key pair error codes

Error	Description
0	Success
41	Failed: Key not found
42	Failed: Deletion error

D.3.6 Crypto support

The hash and asymmetric crypto are specified in the DigSig certificate. The private container pragma crypto is specified in the DDD, which is part of the DigSig certificate.

The crypto support shall be indicated in the Info message.

The minimum crypto signature methods are (see B.4):

- ECDSAwithSHA256 identified as ecdsa-with-SHA256 (see B.4.2)
- NULLwithNULL (see B.4.4)
- NULLwithCRC32 (see B.4.5)
- NULLwithSHA256 identified as sha256 (see B.4.2)
- FP256BNwithSHA256 (see B.4.6)

D.3.7 Configuration

The configuration command configures the response format as required by a connected application for that specific TCP/IP connection, i.e. a configuration is applicable for a single TCP/IP connection. The configuration command has the following format:

```
{"Configure":{<Table D.6 configuration fields>}}
```

Table D.6 — Configuration fields

Field name	Description	Value	Default
Base64	Switch binary return values between `Base64String` and `HexString`	JSON boolean	false
Envelope	Select the envelope format	"RAW" | "URI"	"RAW"

The URI envelope CIDSnip shall use `Base64String` in accordance with 8.3.3.3.

The following applies:

- `{"Configure":{<configuration fields>}}` will set the specific values.
- `{"Configure":{}}` will set the default values.
- `{"Configure":null}` will return the set values.

In all cases the response has the following format:

```
{"Configure":{<configuration fields>}}
```

If the command is in error the response shall be as follows:

```
{"Configure":{"Error":"Bad configuration"}}
```

D.3.8 EncodeGenerate

D.3.8.1 Request format

The DecodeVerify request is stateless and atomic, i.e. the request shall be executed without interruption, and it shall not change the state of the service. As such, the information for the request shall be provided as a single request.

```
{"20248":<DDDdataTagged limited to the leaf fields, i.e. not structjoin'ed>}
```

NOTE 1 The DDDdataTagged contains the DAID and the CID from which the appropriate DigSig certificate is selected for the DigSig encoding and generation.

NOTE 2 The encryption method is obtained from the DigSig certificate.

NOTE 3 The service generates SigData from DDDdataTagged.

D.3.8.2 Response format

```
{"20248":{"ResponseCode":{"Code":<JSON number – code as per Table F.1>,
                          "Desc":<JSON string – error as per Table F.1>},
          "CIDSnip":<CIDSnip binary>,
          "DataSnips":{<DataSnip name>:<DataSnip binary>…}}
```

EXAMPLE With reference to the D.3.9 example.

```
{"20248":{"ResponseCode":{"Code":0,"Desc":"No Error"},
          "CIDSnip":"wJgLT3UaawN5RRiWnEEAA==",
          "DataSnips":{"aa":"jhflkwie7f432","bb":"4j53ds90j42="}}
```

D.3.9 Information notice

The information notice is used by the application to predetermine which DigSigs is of interest and how to obtain the DataSnips of those DigSigs.

The information notice is sent by the service:

- As the first message to the application after the connection was established.
- Unsolicited when the service detects a change to the list of DigSig certificates in the store.
- On request with the command:

  ```
  {"Info":null}
  ```

The information notice field uses the following format:

```
"Info":{"Version":<version as specified in D.3.2>,
        "Crypto":{"Signature":[<signature function>,…],
                  "PrivateContainer":[<private container function>…]},
        "DigSigs":{"<IAC>-<CIN>-<CID>":
```

```
                    {"DAID":"<DAID in Binary>",
                     "CID":<CID number>,
                     "DataSnips":{"<fieldid value>":<pragma value>…}}…}}
```

The allowed pragmas are entertext and readmethod.

EXAMPLES "DataSnips" field values:

```
{"bb":{"entertext":null}}
{"bb":{"readmethod":{"methodname":["ISO/IEC 18000-63"],
       "methodmemory":[3],"methodoffset":128,"methodlength":128}}}
```

EXAMPLE ISO/IEC 18000-63

```
{"Info":
  {"Version":<version as specified in D.3.2>,
   "Crypto":{"Signature":["NULLwithNULL", "NULLwithCRC32", "NULLwithSHA256",
                          "ECBNwithSHA256","FP256BNwithSHA256","ECDSAwithSHA256"]},
   "DigSigs":
    {"QC-FVXX-106":{"DAID":":C098:0B4F:75","CID":106,
                    "DataSnips":{"aa":{"readmethod":{"methodname":["ISO/IEC18000-63"],
                                                     "methodmemory":[3],
                                                     "methodoffset":128,
                                                     "methodlength":128}},
                                 "bb":{"readmethod":{"methodname":["ISO/IEC18000-63"],
                                                     "methodmemory":[2],
                                                     "methodoffset":128,
                                                     "methodlength":128}}}},
     "QC-FVXX-107":{"DAID":":C098:0B4F:75","CID":107,
                    "DataSnips":{"aa":{"readmethod":{"methodname":["ISO/IEC18000-63"],
                                                     "methodmemory":[3],
                                                     "methodoffset":128,
                                                     "methodlength":128}},
                                 "bb":{"readmethod":{"methodname":["ISO/IEC18000-63"],
                                                     "methodmemory":[2],
                                                     "methodoffset":128,
                                                     "methodlength":128}}}}}}
```

NOTE 1 ISO/IEC 18000-63 data configuration and read method considerations are:

- An ISO/IEC 18000-63 tag automatically provides the inventory data. Other data is explicitly read.
- The ISO/IEC 18000-63 inventory data consists of:
  - The protocol control word (PC) that contain the standards toggle bit (the T flag) and the application family identifier (AFI):
    - T = 1 and AFI = 0x92 indicates an ISO data structure with the unique item identifier (UII) encoded as the CIDSnip.
    - T = 0 or 1, AFI ≠ 0x92 and the UserMemory DSFID = 0x11 indicates the CIDSnip follows the data storage format identifier (DSFID) in the UserMemory. The DSFID is the first 8 bits of UserMemory (MB 11).
  - Inventory data (UII).

NOTE 2 With URI data encoding the CIDSnip is the data following the "?" (see 8.3.3.3).

NOTE 3 The CIDSnip follows the ISO/IEC 15434 message structure, Format ‘06’ for ASC MH 10 Data Identifier "6R".

D.3.10 privatecontainer and crypto support

The application shall be responsible for the private container pragma. It is foreseen that the private container handling will be provided as an independent microservice interoperable with this interface specification.

The application shall be responsible for the crypto within a read method. It is foreseen that the crypto handling will be provided as an independent microservice interoperable with this interface specification.

Annex E
(informative)

Typical DigSig DecoderVerifier device architecture

E.1 General

This annex should be read with Annex D.

E.2 Architecture

Figure E.1 illustrates a typical DigSig capable AIDC Reader device architecture. The AIDC Reader device is the device responsible for reading the DigSig from the data carrier(s). It should verify the DigSig. The Application device is a separate device which uses the DigSig. The Application device verifies the DigSig before it is used, even though it has no ability to read data carriers. Both the Application and AIDC Reader devices are DigSig DecoderVerifier devices as specified in this document.

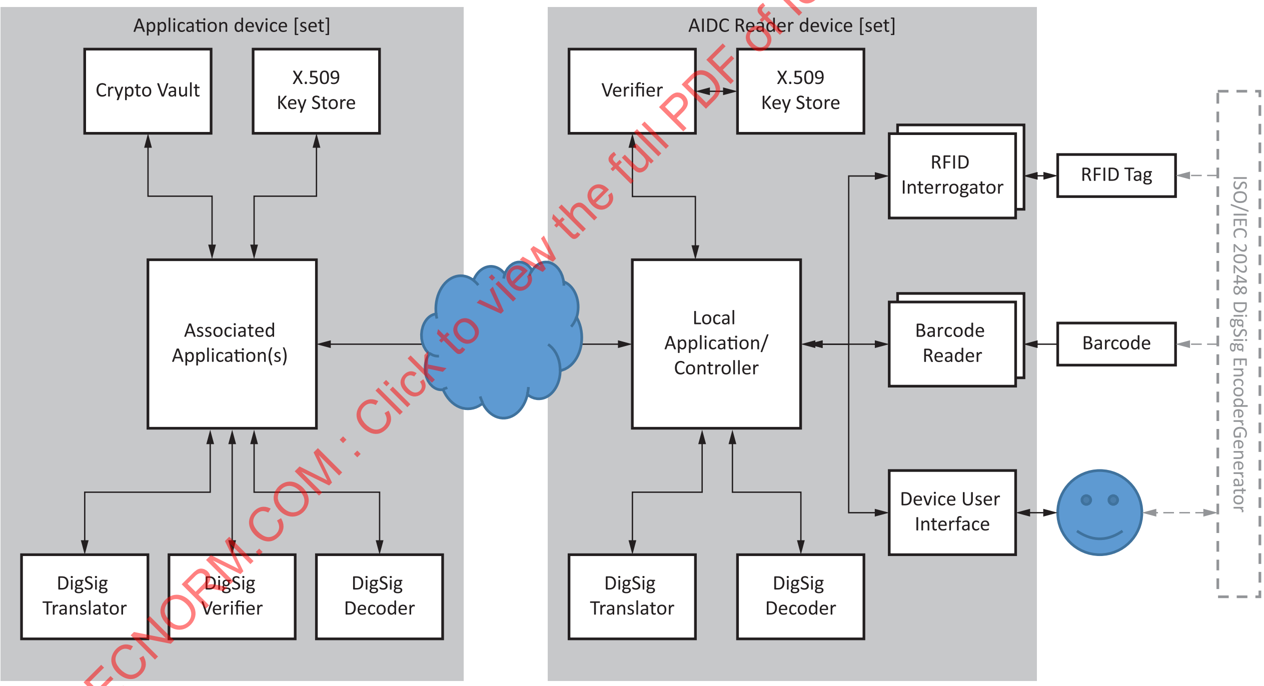


Figure E.1 — Typical DigSig DecoderVerifier use architecture

The architecture specifies the following modules:

- Local Application/Controller: This is an application which runs on a device with the capability to read/interrogate data carriers. It may include an operator interface. This device is typically designed and deployed for a specific purpose.
- Associated Application: This is an application which is connected to the Local Application/Controller with the purpose to receive data carrier read instance information from the Local Application/Controller.
- RFID Interrogators: RFID Interrogators typically interrogate the data carrier providing direct and selective access to memory and over the air security.

- Barcode Readers: Barcode readers typically read a barcode as a single unit. They provide the data length and encoding information allowing the Application to interpret the data easily.
- Device User Interface: This interface is typically associated with a handheld device but can be a console for an automated portal. It displays information and allows text input.
- X.509 Key Store: The X.509 Key Store is typically a standards-based component of the operating system. It validates and stores X.509 certificates as provided by a trusted application. Content of the X.509 Key Store is not necessarily secret but can be considered authentic because it is digitally signed. However, access to it should be strictly controlled to prevent the insertion of erroneous certificates. DigSig implementations typically use an independent instance of the store local to the application providing isolation between the DigSig certificates and other certificates stored on the device.
- Crypto Vault: The Crypto Vault is typically a dedicated hard or software module which performs specific cryptographic functions. The Crypto Vault is used to decrypt private containers as directed by the `privatecontainer` directive. It may also be used with cryptographic air protocols as directed by the `readmethod`. It normally also acts as a secret store for keys (notably symmetric keys and the private key of a key pair) and has an ability to securely obtain such keys based on a directive. Various commercial products are available which are, e.g. used in POS terminals and ATMs.
- DigSig Decoder: The DigSig Decoder module decodes an array of Snips into DDDdata.

  DataSnips containing a `privatecontainer` may be decoded by the Application and before being passed to the DigSig decoder. Such decoded DataSnips should be marked as having been decrypted allowing the DigSig Decoder to decode the fields of the `privatecontainer`. The DigSig decoder should assume a `privatecontainer` DataSnip is in its encrypted form unless instructed otherwise.

  The DigSig Decoder should be able to indicate, using the CIDSnip and the selected DDD, if and where DataSnips can be obtained from.
- DigSig Verification Module: The DigSig Verifier module verifies DDDdata using the referenced DigSig certificate. It generates SigData from the DDDdata.
- DigSig Translation Module: The DigSig Translation Module uses `displayformat` to generate DDDdataDisplay from DDDdata.

The process flow is as follows:

a) The Local Application/Controller reads/interrogates a CIDSnip.

b) The Local Application/Controller extracts the CID from the CIDSnip.

c) The Local Application/Controller selects the most appropriate DigSig certificate for the decoding from the X.509 Key Store. If not available, it searches for it according to its local knowledge.

d) The Local Application/Controller extracts the DDD from the DigSig certificate and passes it along with the CIDSnip to the DigSig Decoder.

e) The DigSig Decoder decodes the CIDSnip into DDDdata. If DataSnips are needed to complete the DDDdata the DigSig Decoder returns a DataSnip map containing instructions on how to obtain data.

f) The Local Application/Controller reads all the required DataSnips (`privatecontainer` DataSnips are not decrypted at this stage). It then passes the array of Snips to the decoder for decoding.

g) When the DDDdata are correctly decoded, the Local Application/Controller provides the DigSig certificate and the DDDdata to the DigSig Verifier for verification.

h) If the DigSig verification is accepted, the Local Application/Controller may request the DigSig Translator to generate DDDdataDisplay and DDDdataTagged for its own purposes.

i) The Local Application/Controller may, with proper authority, decrypt `privatecontainer` DataSnips for them to be decoded into the DDDdata. DDDdataDisplay and DDDdataTagged then contain the fields of the `privatecontainer`.

j) The Local Application/Controller may pass the DDDdata to Associated Applications who can perform their own verification without the need to read/interrogate the data carrier.

E.3 Example interface

E.3.1 Principles

This example DigSig DecoderVerifier interface is widely used to provide a TCP/IP command-response (message) service. It is used to decode and verify DigSigs read by Automated Identification Data Capture (AIDC) devices.

All commands shall be responded to and processed in sequence.

E.3.2 Version

See D.3.2.

E.3.3 Interface message

E.3.3.1 Method

See D.3.3.1.

E.3.3.2 Message framing

See D.3.3.2

E.3.3.3 Binary values

See 4.4.

E.3.3.4 Communication start-up

See D.3.3.3.

E.3.3.5 Communication termination

See D.3.3.5.

E.3.3.6 Message error

See D.3.3.6.

E.3.4 Certificate store

The certificate store contains the desired 20248 certificates and revocation lists.

An independent trusted service should manage this store.

The DecoderVerifier shall watch the store for a change on which it shall enact the change and notify connected applications of the change.

E.3.5 DigSig certificates and revocation lists

E.3.5.1 General

A DecoderVerifier should be able to obtain DigSig certificates, certificate revocation list and DigSig revocation lists. The application may perform these functions in the following manners:

- DigSig certificates: The application obtains and provides the DecoderVerifier with the DigSig Certificates for its targeted DigSig AIDC data.
- Certificate revocation: The application checks or receives certificate revocation lists where after it notifies the DecoderVerifier of the certification status.
- DigSig revocation list: The application obtains the DigSig revocation lists (see Annex J) and provides it as is to the DecoderVerifier.

E.3.5.2 Add a DigSig certificate

See D.3.5.3

E.3.5.3 Delete a DigSig certificate

See D.3.5.4

E.3.5.4 Change a DigSig certificate revocation status

The following command shall be used to change a DigSig certificate status:

```
{"CertRevocationStatus":{"DAID":<DAID>,"CID":<CID>,"Revoked":<true or false>}}
```

NOTE 1 The default status of a certificate is not revoked (`false`).

NOTE 2 A certificate can only be suspended.

The response has the following format:

```
{"CertRevocationStatus":{"ResponseCode":{"Code":<JSON number – code as per Table E.1>,
                                         "Desc":<JSON string – error as per Table E.1>}}}
```

Table E.1 — Change revocation status error codes

Error	Description
0	Success
31	Failed: Cert not found
32	Failed: Storage error

E.3.5.5 Add a DigSig revocation list

The following command shall be used to add a DigSig revocation list:

```
{"AddDigSigRL":<A DigSig revocation list as specified in J.3.4>}
```

The response has the following format:

```
{"AddDigSigRL":{"ResponseCode":{"Code":<JSON number – code as per Table E.2>,
                                "Desc":<JSON string – error as per Table E.2>}}}
```

Table E.2 — Add a DigSig revocation list errors

Error	Description
0	Success

Table E.2 *(continued)*

Error	Description
41	Failed: Associated {DAID, CID} not found
42	Failed: Revocation list format error
43	Failed: Storage error

E.3.5.6 Delete a DigSig revocation list

The following command shall be used to delete a DigSig revocation list:

```
{"DelDigSigRL":{"DAID":<DAID>,"CID":<CID>}}
```

The response has the following format:

```
{"DelDigSigRL":{"ResponseCode":{"Code":<JSON number – code as per Table E.3>,
                                "Desc":<JSON string – error as per Table E.3>}}}
```

Table E.3 — Delete a DigSig revocation list error codes

Error	Description
0	Success
51	Failed: List not found
52	Failed: Deletion error

E.3.6 Crypto support

See D.3.6.

E.3.7 Configuration

The configuration command configures the response format as required by a connected application for that specific TCP/IP connection, i.e. a configuration is applicable for a single TCP/IP connection. The configuration command has the following format (the rules of D.3.7 apply):

```
{"Configure":{<Table E.4 configuration fields>}}
```

Table E.4 — Configuration fields

Field name	Description	Value	Default
DDDdata	Return DDDdata	JSON boolean	false
SigData	Set return SigData	JSON boolean	false
DDDdataTagged	Set return DDDdataTagged	JSON boolean	true
DDDdataDisplay	Set return DDDdataDisplay	JSON boolean	false
StructuredDoc	Set return StructuredDoc	JSON string with Media Type (IEFT RFC 6838), see Annex O	false
Timezone	Set the time zone	JSON string: ISO 8601 time zone	Device time zone
DateTime	Set the time	JSON string: ISO 8601 format YYYY-MM-DDThh:mm:ss.sssZ No time zone means local time.	Device time
Language	Set the language for DDDdataDisplay	JSON string: See 8.2.5.	First in list

E.3.8 DecodeVerify request

E.3.8.1 Request format

The DecodeVerify request is stateless and atomic, i.e. the request shall be executed without interruption, and it shall not change the state of the service. As such, the information for the request shall be provided as a single request. This also applies where more data has been requested to complete a verification.

```
{"20248":{"CIDSnip":"<binary data>",
          "DataSnips":{"<field name>":"<binary data>"…}}}
```

E.3.8.2 Response format

```
{"20248":{"ResponseCode":{"Code":<JSON number - code as per Table F.1>,
                          "Desc":<JSON string - error as per Table F.1>},
          "MissingDataSnips":[<list of DataSnip root field names>],
          "DDDdata":[<DDDdata>],
          "SigData":[<SigData>],
          "DDDdataTagged":{<DDDdataTagged>},
          "DDDdataDisplay":{<DDDdataDisplay>},
          "StructuredDoc":<Media Type (IEFT RFC 6838) URI>}}
```

`MissingDataSnips` is omitted when all the DataSnips are complete.

`DDDdata`, `SigData`, `DDDdataTagged` and `DDDdataDisplay` are included by configuration.

The DigSig DecoderVerifier should decode and return field values for which it has data as contained in a Snip. Fields without data are returned with a null value.

E.3.9 Information notice

See D.3.9.

E.3.10 privatecontainer and crypto support

See D.3.10.

Annex F
(normative)

DigSig error codes

The DigSig EncoderGenerator (G) and DecoderVerifier (V) error codes are provided in Table F.1.

Table F.1 — DigSig error codes

Code	Applicable	Error	Notes
0	V	DigSig verification accepted; No error	
1	V	DigSig verification accepted; DigSig certificate revocation not checked	The local application may be configured not to check for a certificate revocation. This check may be performed for each verification or periodically.
2	V	DigSig verification accepted; DigSig certificate revoked	
3	V	DigSig verification accepted; DigSig revocation not checked	The local application may be configured not to check for a certificate revocation. This check may be performed for each verification or against a revocation list obtained during another process.
4	V	DigSig verification accepted; DigSig revoked	
5	V	DigSig verification accepted; DigSig certificate expired	The verification is accepted if the certificate expiry is ignored.
6	V	DigSig verification rejected	The cryptographic function rejected the verification.
7	GV	DigSig certificate expired	The DigSig certificate is expired, as such no cryptographic verification was attempted.
8	GV	DigSig certificate is rejected	The DigSig certificate is cryptographically rejected.
9	V	DigSig certificate trust chain is rejected	The DigSig certificate trust chain is cryptographically rejected.
10	V	DigSig certificate revoked	The DigSig certificate is revoked.
11	GV	DigSig certificate format error	The DigSig certificate cannot not be decoded.
12	V	DigSig certificate not local	The CID could not be matched with a locally stored DigSig certificate.
13	V	DigSig certificate not found online	The CID could not be found at the locally configured DAs.
14	V	DigSig certificate not available	The network resource is not available preventing on online search.
15	GV	DDD format error	The DDD could not be decoded.
16	GV	DDDdata does not match the DDD	
17	V	CIDSnip incomplete	The CID could not be decoded from the CIDSnip.
18	V	DataSnip incomplete	The DataSnip requires more bits/text to be decoded according to the DDD.
19	V	Snips incomplete	More Snips are required to complete a DigSig decoding.
20	V	Read method not available	
21	V	Read method error	

Annex G
(informative)

Digital Signature use considerations

Digital signatures within the public key infrastructure (PKI) and web-of-trust (WOT) concepts are widely used and supported by most reputable operating systems. It is important to note that digital signatures methods may be used without a PKI or WOT when it is used within a trusted closed system or service.

A familiarity with recognized or standardized methods, e.g. FIPS PUB 186-4 and IEEE P1363, is recommended.

A digital signature scheme typically consists of three algorithms.

- A key generation algorithm that selects a private-public key pair uniformly at random from a set of possible keys. The algorithm outputs the private key and a corresponding public key. The private key remains secret. The public key is published in a digital certificate.
- A signing algorithm (the hash and cryptographic functions) that, given a message and a private key, produces a signature.
- A signature verification algorithm, which includes the hash and cryptographic functions, that, when provided with a message, public key, and a signature, either accepts or rejects the claim of authenticity of the message.

A successful digital signature scheme has the following properties:

- It should be computationally infeasible to generate a valid digital signature if a party does not possess the private key.
- The digital signature which contains the public key and information to read, decode and verify a digital signature should be verifiable and traceable.

Annex H
(informative)

Example of a DigSig certificate

This annex gives an example of a DigSig certificate to indicate where the DDD component is placed.

NOTE This example DigSig certificate uses the information from Annex I.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = digsig.tools
        Validity
            Not Before: Jan 25 06:49:59 2021 GMT
            Not After : Jan 25 06:49:59 2041 GMT
        Subject: C = DE, O = Sieveo, CN = https://www.dept-edu.com/daid/QC%20DGSG/cid/110
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:2a:3e:93:e6:7b:61:88:90:91:ac:9c:33:39:57:
                    04:ef:ce:ee:06:e0:c4:86:3e:74:29:7a:a3:4d:27:
                    9d:a5:fd:1d:8b:db:6c:53:00:25:e7:45:36:da:99:
                    8b:80:7b:ce:fd:32:e7:49:b8:63:7a:78:7f:ae:83:
                    2d:61:cf:34:d7
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            1.0.20248:
                {"digsiginfo":{"specificationversion":"ISO/IEC 20248:2022","dauri":"https:
                //www.dept-edu.com","daid":"QC DGSG","cid":110,"verificationurii":["https:
                //www.dept-edu.com"],"revocationuri":["https://www.dept-edu.com/revoked"],
                "preverify":{"en":"Inspect the certificate holder's original ID Document",
                "af":"Inspekteer die sertifkaathouer se oorspronklike ID Document"},"accep
                tverify":{"en":"This is a valid Department of Education Certificate.","af"
                :"Geldige Departement van Onderwys Sertifikaat."},"rejectverify":{"en":"Th
                is is not a valid Department of Education Certificate.","af":"Die Sertifik
                aat is nie 'n geldig Departement van Onderwys Sertifikaat nie."},"postveri
                fy":{"en":"For more information contact the Department of Education.","af"
                :"Vir meer inligting kontak die Departement van Onderwys."},"structureddoc
                uri":{"en":"https://www.dept-edu.com/text/plain/uni-cert-en","af":"https:/
                /www.dept-edu.com/text/plain/uni-cert-af"}},"datafields":[{"fieldid":"spec
                ificationversion"},{"fieldid":"dauri"},{"fieldid":"daid"},{"fieldid":"cid"
                },{"fieldid":"signature"},{"fieldid":"timestamp","range":"[2022-01-01T00:0
                0:00..2024]"},{"fieldid":"studentname","type":"string","fieldname":{"en":"
                Student Name","af":"Student Naam"},"description":{"en":"Name of the person
                 who holds this certificate","af":"Die naam van die persoon aan wie die se
                rtifikaat behoort"}},{"fieldid":"idnumber","type":"string","fieldname":{"e
                n":"ID Number","af":"ID Nommer"},"description":{"en":"Scan the ID Document
                 barcode","af":"Lees the ID Document strepieskode"},"pragma":{"readmethod"
                :{"methodname":["ISO/IEC 15417"]}}},{"fieldid":"years","type":"isodate","r
                ange":"[2021..2024]","fieldname":{"en":"Years registered","af":"Jare gereg
                istreed"},"cardinality":"{1,5}"},{"fieldid":"coursename","type":"string","
                fieldname":{"en":"Course Name","af":"Kursus Naam"}},{"fieldid":"department
                ","type":"string","fieldname":{"en":"Department","af":"Department"}},{"fie
                ldid":"departmentURL","type":"string","fieldname":{"en":"Department URL","
                af":"Department URL"},"bsign":false},{"fieldid":"date","type":"isodate","r
                ange":"[2021-01-01..+4]","fieldname":{"en":"Certificate Date","af":"Sertif
                ikaat Datum"}},{"fieldid":"subjectgrades","type":"struct","fieldname":{"en
                ":"Subject Grades","af":"Vak Punt"},"cardinality":"{1,5}","fields":[{"fiel
                did":"subjectname","type":"string","fieldname":{"en":"Subject Name","af":"
                Vak Naam"}},{"fieldid":"subjectPurpose","type":"enum","enumvalues":["degre
                e","extra"],"enumvaluedesc":[{"en":"Degree","af":"Graad"},{"en":"Extra","a
                f":"Ekstra"}]},{"fieldid":"grade","type":"enum","fieldname":{"en":"Grade",
                "af":"Punt"},"enumvalues":["A","B","C","D","E"]}]}]}
    Signature Algorithm: sha256WithRSAEncryption
         85:f3:d5:bc:81:40:d1:84:df:f4:5e:94:00:0d:78:64:62:f2:
         80:6b:44:c6:16:14:08:ec:ff:d1:d5:04:ea:1e:5e:d4:58:a2:
         72:20:0d:f9:aa:0d:23:b6:a5:04:d5:c2:88:78:08:f4:fc:dc:
         86:77:03:7f:64:4a:51:65:d2:f6:f4:c4:c1:43:e6:bf:eb:5a:
         08:76:52:6e:da:34:aa:d1:e6:99:da:5c:31:26:db:57:1d:e4:
         a4:0c:0d:02:9f:cd:81:8d:4c:b6:7d:a8:04:eb:cd:41:91:8f:
         e9:cf:a4:83:05:a4:7d:41:0e:ee:37:0d:71:34:a4:33:b9:fb:
         17:fb:8e:69:e4:f0:4f:3b:14:ef:c3:ad:14:38:7d:3e:9a:6a:
         ae:06:03:f0:bf:17:0d:b1:7f:e5:9b:1a:b1:82:96:76:bd:e4:
         54:ce:43:c3:0a:85:b4:47:c4:09:3a:45:98:35:d9:ca:e8:c2:
         ae:e4:64:66:be:12:d9:6b:88:47:c2:fb:9e:f7:90:7d:29:b3:
         c3:15:7c:ae:ae:59:57:34:ba:19:3a:e7:79:34:8f:15:22:27:
         0a:29:3e:d6:02:e1:65:dc:4a:d5:fc:df:2a:be:cc:be:8b:89:
         bc:b8:5c:dd:b8:4a:2b:54:72:b0:8e:6d:97:00:b0:c6:09:2e:
         53:e6:c0:4f
```

Annex I
(informative)

Example DDD for a physical certificate

I.1 General

This example illustrates the various DDD data constructs. It is not meant to be representative of either the standard use or a university certificate.

ABC University issues certificates of conformance with its courses to its successful students. To verify a certificate, the verifier should also verify the identity of the certificate holder by scanning the linear barcode on his/her national identification document. In this case, the certificate shall be presented for verification with the identity document of the certificate holder. This method assists in the positive identification of the certificate holder. A comment line in the DDD alerts the verifier to this requirement.

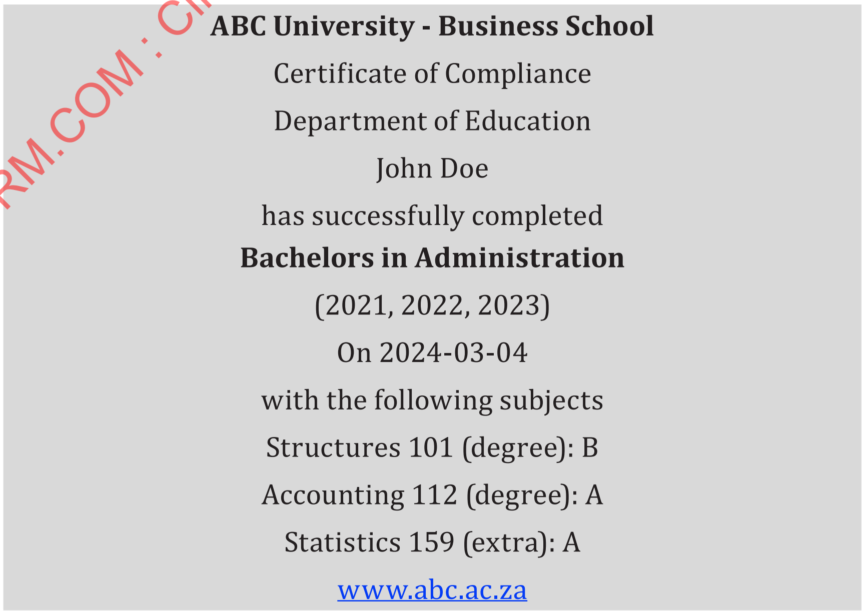
Two data carriers are used:

— The first data carrier contains the information of the certificate excluding the certificate holder's identity number. It also contains a URI for the university, which is not part of the data to be verified but is included in the DigSig.

This data carrier may be a 2D barcode, ISO/IEC 18000-63 or an NFC tag. It uses the URI envelope to support online verification typically with a standard mobile phone.

— The second data carrier is the linear barcode on the South African national identification book containing the identification number of the certificate holder. This DataSnip is a TEXT encoded.

I.2 Example university course certificate



ABC University - Business School

Certificate of Compliance

Department of Education

John Doe

has successfully completed

Bachelors in Administration

(2021, 2022, 2023)

On 2024-03-04

with the following subjects

Structures 101 (degree): B

Accounting 112 (degree): A

Statistics 159 (extra): A

www.abc.ac.za

I.3 Example DDD

```
{"digsiginfo":
  {"specificationversion":"ISO/IEC 20248:2022",
    "dauri":"https://www.dept-edu.com",
    "daid":"QC DGSG","cid":110,
    "verificationuri":["https://www.dept-edu.com"],
    "revocationuri":["https://www.dept-edu.com/revoked"],
    "preverify":
     {"en":"Inspect the certificate holder's original ID Document",
      "af":"Inspekteer die sertifkaathouer se oorspronklike ID Document"},
    "acceptverify":
     {"en":"This is a valid Department of Education Certificate.",
      "af":"Geldige Departement van Onderwys Sertifikaat."},
    "rejectverify":
     {"en":"This is not a valid Department of Education Certificate.",
      "af":"Die Sertifikaat is nie 'n geldig Departement van Onderwys Sertifikaat nie."},
    "postverify":
     {"en":"For more information contact the Department of Education.",
      "af":"Vir meer inligting kontak die Departement van Onderwys."},
    "structureddocuri":
     {"en":"https://www.dept-edu.com/text/plain/uni-cert-en",
      "af":"https://www.dept-edu.com/text/plain/uni-cert-af"}},
  "datafields":
   [{"fieldid":"specificationversion"},
    {"fieldid":"dauri"},
    {"fieldid":"daid"},
    {"fieldid":"cid"},
    {"fieldid":"signature"},
    {"fieldid":"timestamp","range":"[2022-01-01T00:00:00..2024]"},
    {"fieldid":"studentname","type":"string",
     "fieldname":{"en":"Student Name","af":"Student Naam"},
     "description":
      {"en":"Name of the person who holds this certificate",
       "af":"Die naam van die persoon aan wie die sertifikaat behoort"}},
    {"fieldid":"idnumber","type":"string",
     "fieldname":{"en":"ID Number","af":"ID Nommer"},
     "description":
      {"en":"Scan the ID Document barcode",
       "af":"Lees the ID Document strepieskode"},
     "pragma":{"readmethod":{"methodname":["ISO/IEC 15417"]}}},
    {"fieldid":"years","type":"isodate","range":"[2021..2024]",
     "fieldname":{"en":"Years registered","af":"Jare geregistreed"},
     "cardinality":"{1,5}"},
    {"fieldid":"coursename","type":"string",
     "fieldname":{"en":"Course Name","af":"Kursus Naam"}},
    {"fieldid":"department","type":"string",
     "fieldname":{"en":"Department","af":"Department"}},
    {"fieldid":"departmentURL","type":"string",
     "fieldname":{"en":"Department URL","af":"Department URL"},
     "bsign":false},
    {"fieldid":"date","type":"isodate","range":"[2021-01-01..+4]",
     "fieldname":{"en":"Certificate Date","af":"Sertifikaat Datum"}},
    {"fieldid":"subjectgrades","type":"struct",
     "fieldname":{"en":"Subject Grades","af":"Vak Punt"},
     "cardinality":"{1,5}",
     "fields":
      [{"fieldid":"subjectname","type":"string",
        "fieldname":{"en":"Subject Name","af":"Vak Naam"}},
       {"fieldid":"subjectPurpose","type":"enum",
        "enumvalues":["degree","extra"],
        "enumvaluedesc":[{"en":"Degree","af":"Graad"},{"en":"Extra","af":"Ekstra"}]},
       {"fieldid":"grade","type":"enum",
        "fieldname":{"en":"Grade","af":"Punt"},
        "enumvalues":["A","B","C","D","E"]}]}]}
```

I.4 Example Snips

The example DDD and data set results in the following Snips:

CIDSnip in hex encoding.

```
0xC098099640006EE100EF28DC6B3A36E4B77869BB6DFF75B6E5C79B7985B11262842A738A7424ED60E563
C1D094DED0DC4088DECB61B189919181C9A1C9C1C981946361B42616368656C6F727320696E2041646D696
```

```
E697374726174696F6EA99195C185C9D1B595B9D2D6465706172746D656E7455524C6648EE537472756374
75726573203130311B9058D8DBDD5B9D1A5B99C80C4C4C82E537461746973746963732031353 98
```

CIDSnip in base64url encoding.

```
wJgJlkAAbuEA7yjcazo25Ld4abtt_3W25cebeYWxEmKEKnOKdCTtYOVjwdCU3tDcQIjey2GxiZGRgcmhycHJgZ
RjYbQmFjaGVsb3JzIGluIEFkbWluaXN0cmF0aW9uqZGVwYXJ0bWVudLWRlcGFydG1lbnRVUkxmSO5TdHJ1Y3R1
cmVzIDEwMRuQWNjb3VudGluZyAxMTILlN0YXRpc3RpY3MgMTU5g
```

DataSnip: `612209498902` as a linear barcode using ISO/IEC 15417.

The URI envelope:

```
https://www.dept-edu.com?wJgJlkAAbuEA7yjcazo25Ld4abtt_3W25cebeYWxEmKEKnOKdCTtYOVjwdCU3tDc
QIjey2GxiZGRgcmhycHJgZRjYbQmFjaGVsb3JzIGluIEFkbWluaXN0cmF0aW9uqZGVwYXJ0bWVudLWRlcGFydG1lbn
RVUkxmSO5TdHJ1Y3R1cmVzIDEwMRuQWNjb3VudGluZyAxMTILlN0YXRpc3RpY3MgMTU5g
```

I.5 Example DDDdata$^{\text{input}}$

```
["ISO/IEC 20248:2022","https://www.dept-edu.com","QC DGSG",110,
 null,
 null,
 "John Doe","612209498902",["2021","2022","2023"],
 "Bachelors in Administration","department","www.abc.ac.za","2024-03-04",
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]]]
```

I.6 Example DDDdata

```
["ISO/IEC 20248:2022","https://www.dept-edu.com","QC DGSG",110,
 ":EF28:DC6B:3A36:E4B7:7869:BB6D:FF75:B6E5:C79B:7985:B112:6284:2A73:8A74:24ED:60E5",
 "2021-11-04T08:00:00",
 "John Doe","612209498902",["2021","2022","2023"],
 "Bachelors in Administration","department","www.abc.ac.za","2024-03-04",
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]]]
```

I.7 Example SigData

This shows all the fields as specified by the DDD with `bsign` set to true, which will be signed.

```
["ISO/IEC 20248:2022","https://www.dept-edu.com","QC DGSG",110,"2024-11-04T08:00:00","John
Doe","612209498902",["2021","2022","2023"],"Bachelors in Administration","Business
School","2024-03-04",[["Structures 101","degree","B"],["Accounting
112","degree","A"],["Statistics 159","extra","A"]]]
```

SigData does NOT contain whitespaces or any other formatting. This below example is formatted for readability and comparison purposes.

```
["ISO/IEC 20248:2022","https://www.dept-edu.com","QC DGSG",110,
 "2024-11-04T08:00:00",
 "John Doe","612209498902",["2021","2022","2023"],
 "Bachelors in Administration","Business School","2024-03-04",
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]]]
```

I.8 Example DDDdataTagged

```
{"specificationversion":"ISO/IEC 20248:2022",
 "dauri":"https://www.dept-edu.com",
 "daid":"QC DGSG",
 "cid":110,
 "signature":
  ":EF28:DC6B:3A36:E4B7:7869:BB6D:FF75:B6E5:C79B:7985:B112:6284:2A73:8A74:24ED:60E5",
 "timestamp":"2024-11-04T08:00:00",
 "studentname":"John Doe",
 "idnumber":"612209498902",
 "years":["2021","2022","2023"],
 "coursename":"Bachelors in Administration",
 "department":"Business School",
 "departmentURL":"www.abc.ac.za",
 "date":"2024-03-04",
 "subjectgrades":
  [{"subjectname":"Structures 101","subjectPurpose":"degree","grade":"B"},
   {"subjectname":"Accounting 112","subjectPurpose":"degree","grade":"A"},
   {"subjectname":"Statistics 159","subjectPurpose":"extra","grade":"A"}]}
```

I.9 Example DDDdataDisplay

This example requested English as the display language.

```
{"digsiginfo":
  {"specificationversion":"ISO/IEC 20248:2022",
   "dauri":"https://www.dept-edu.com",
   "daid":"QC DGSG",
   "cid":110,
   "verificationuri":"https://www.dept-edu.com/verify",
   "revocationuri":"https://www.dept-edu.com/revoke",
   "preverify":"Inspect the certificate holder's original Id Document",
   "acceptverify":"This is a valid Department of Education Certificate.",
   "rejectverify":"This is not a valid Department of Education Certificate.",
   "postverify":"For more information contact the Department of Education.",
   "structureddocuri":"https://www.dept-edu.com/uni-cert"},
 "datafields":
  [{"fieldid":"specificationversion",
    "displayvalue":"ISO/IEC 20248:2022"},
   {"fieldid":"dauri",
    "displayvalue":"https://www.dept-edu.com"},
   {"fieldid":"daid",
    "displayvalue":"QC DGSG"},
   {"fieldid":"cid",
    "displayvalue":"110"},
   {"fieldid":"signature",
    "displayvalue":
     ":EF28:DC6B:3A36:E4B7:7869:BB6D:FF75:B6E5:C79B:7985:B112:6284:2A73:8A74:24ED:60E5"},
   {"fieldid":"timestamp",
    "displayvalue":"2024-11-04T08:00UTC"},
   {"fieldid":"studentname",
    "fieldname":"Student Name",
    "description":"Name of the person who holds this certificate",
    "displayvalue":"John Doe"},
   {"fieldid":"idnumber",
    "fieldname":"ID Number",
    "description":"Scan the ID Document barcode",
    "displayvalue":"612209498902"},
   {"fieldid":"years",
    "fieldname":"Years registered",
    "displayvalue":["2021","2022","2023"]},
   {"fieldid":"coursename",
    "fieldname":"Course Name",
    "displayvalue":"Bachelors in Administration"},
   {"fieldid":"department",
    "fieldname":"Department",
    "displayvalue":"Business School"},
   {"fieldid":"departmentURL",
    "fieldname":"Department URL",
    "displayvalue":"www.abc.ac.za"},
   {"fieldid":"date",
    "fieldname":"Certificate Date",
    "displayvalue":"2024-03-04"},
   {"fieldid":"subjectgrades",
    "fieldname":"Subject Grades",
    "displayvalue":
     [[{"fieldid":"subjectname",
        "fieldname":"Subject Name",
  "displayvalue":"Structures 101"},
 {"fieldid":"subjectPurpose",
  "displayvalue":"Degree"},
 {"fieldid":"grade",
  "fieldname":"Grade",
  "displayvalue":"B"}],
[{"fieldid":"subjectname",
  "fieldname":"Subject Name",
  "displayvalue":"Accounting 112"},
 {"fieldid":"subjectPurpose",
  "displayvalue":"Degree"},
 {"fieldid":"grade",
  "fieldname":"Grade",
  "displayvalue":"A"}],
[{"fieldid":"subjectname",
  "fieldname":"Subject Name",
  "displayvalue":"Statistics 159"},
 {"fieldid":"subjectPurpose",
  "displayvalue":"Extra"},
 {"fieldid":"grade",
  "fieldname":"Grade",
  "displayvalue":"A"}]]}]}
```

Annex J
(normative)

DigSig revocation specifications

J.1 General

J.1.1 DigSig revocation mechanism

A Domain Authority DigSigs may revoke

- a DigSig certificate, which will result in all the DigSigs referencing this revoked DigSig certificate to be revoke, and
- a specific DigSig.

The methods and processes for revocation of certificates are outside the scope of this document. Best practice guidelines and standard methods are well known within the X.509 application domain. The same applies for the distribution of certificate revocation lists and the checking of such lists. This annex concerns itself with

- DigSig revocation lists, and
- the online checking of the revocation status of a DigSig.

Revocation and the revocation list period should be carefully considered based on the required level of trust of the authenticity within the DigSig use environment.

NOTE 1 The need to check in real-time the revocation status of DigSigs counteracts the benefits of offline verification. This real-time online check, as with all cloud-based services, can be vulnerable to denial-of-service attacks. Best practices therefore suggest the use of revocation lists which are periodically distributed.

For ease of use and interoperability the DigSig revocation lists shall use DigSigs with the DigSig Data Descriptions (DDDs) as specified in J.3.3.

NOTE 2 The revocation DigSig certificate and DigSigs are issued and used in the same way than all other DigSig certificates and DigSigs. A Domain Authority will typically issue annually a revocation list DigSig certificate which is used during the year to generate and verify all the revocation lists of the Domain Authority.

To counter denial of service attacks a Domain Authority should distribute the signed revocation lists to trusted proxies and repositories. AIDC applications should obtain these lists periodically to maintain reliable offline operations.

J.1.2 DigSig revocation process

DigSig revocation checking is a subtask of DigSig verification. Neither need to be done when the AIDC data carrier is read, however, the DigSig validity should be checked when (or just before) the DigSig is used. The risk in the use of the DigSig determines when a revocation check is done, if at all. Revocation checking is only failsafe when it is done in real-time, which is often not attainable or efficient. Common revocation practices therefore provide for the creation and distribution of revocation lists, which is used for local revocation checking.

A DigSig may be verified, and its revocation status checked

- by the AIDC reader,
- by the local AIDC application,