INTERNATIONAL STANDARD

ISO/IEC 20248

First edition
2018-03

Information technology — Automatic identification and data capture techniques — Data structures — Digital signature meta structure

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

Reference number
ISO/IEC 20248:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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 document 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).

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).

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

For an explanation on 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 the following URL: www.iso.org/iso/foreword.html.

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

Introduction

This document specifies a “language” which is used to specify data constructs with; how the data constructs can be read from one or more AIDC; and how to decode and verify such data.

This document is an ISO/IEC 9594-8 (Public Key Infrastructure: digital signatures and certificates) application specification for automated identification services. Data capacity and/or data transfer capacity of Automated Identification Data Carriers are limited. This restricts the normal use of a digital signature as specified in ISO/IEC 9594-8 within automated identification services.

This document specifies an effective and interoperable method to specify, read, decode and verify data stored in automated identification data carriers, 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 automated identification data carriers.

The user of this document may use any suitable hashing and asymmetric cryptography method. The choice of cryptography method should be considered carefully and it is advised that only internationally recognized or standardized methods, for example 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 transport applications rely on a secure non-transferable unique identifier to ensure that the data are bound to the tag and/or the vehicle. For such functionality, please refer to ISO/IEC 29167. This specification provides a mechanism to ensure the integrity and authenticity of the data themselves in order to protect against alterations or insertion of false data into the system. It does not provide any means to defend against replay attacks. Including the secure non-transferable unique identifier of a tag, as signed data, allows for the unrefutable link between the tag and the data and provides a means to determine if the data were read from the tag. The reader can place the read DigSig in another DigSig, effectively signing the read transaction. A third party can then verify that the read transaction happened at a given place and time, as well as the data read.

Information technology — Automatic identification and data capture techniques — Data structures — Digital signature meta structure

1 Scope

This document is an ISO/IEC 9594-8 (Public Key Infrastructure: 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. It is worth noting that 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.

A successful verification of the DigSig signifies the following:

— 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 identifier of the data carrier is included in the DigSig it contains, then the data stored on the data carrier can be considered as the original issued copy of the data; the secure identifier will be able to guarantee that the data carrier is authentic.

This document does not specify

— cryptographic methods, nor

— key management methods.

This document is used in conjunction with standard risk assessments of the use environment.

2 Normative references

The following documents are referred to in the text in such a way that some or all of 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/IEC 8824-1[1], *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation — Part 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 9899, *Information technology — Programming languages — C*

ISO/IEC 18004, *Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification*

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

IETF 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 following terms and definitions apply.

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

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

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

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
singular continuous bit or text stream portion of a Data Carrier transmission which contains the CID as the first part

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

[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.

3.4
Data Carrier
device used to store data as a relay mechanism in an AIDC system

EXAMPLE Barcodes, RFID tags and even human memory.

3.5
Data Carrier construct rule
process to prepare the DigSig Envelope for encoding in a particular Data Carrier

3.6
DataSnip
singular continuous bit or text stream portion of a Data Carrier transmission containing data for DDD fields

3.7
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.

3.8
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.

3.9
DigSig
data construct assembled according to this document which contains verifiable information obtained from one or more AIDC

3.10
DigSig Envelope
envelope
data construct assembled according to this document by the EncoderGenerator

3.11
Domain Authority
entity, operating as a trusted third party, responsible for the Digital Signature integrity of a jurisdiction

3.12
integrity
reliability of data that are as they were created according to the required verification parameters

3.13
jurisdiction
independent domain of control in terms of the business or legal (or both) scope of the parties concerned

EXAMPLE Independent countries, separate ministries or departments of a government, or independent companies each with their own legal or business (or both) framework.

3.14
nibble
four-bit aggregation

3.15
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

3.16
protocol
communication specification

3.17
Public Key
key that is publicly available and is used to verify data that were encrypted by its associated Private Key

3.18
Snip
singular continuous bit or text stream portion of a Data Carrier transmission

3.19
Time Zone
time zone code

Note 1 to entry: See ISO 8601.

3.20
UTF-64
64 bit variable-width encoding

Note 1 to entry: See ISO 10646.

3.21
WORD
media physical memory grouping of bits

4 Field and data definitions, abbreviations and symbols

4.1 Field and data definitions

Field and data objects are defined in Clause 8.

4.2 Abbreviations

AFI	Application Family Identifier
AIDC	Automatic Identification Data Carrier
AutoID	Automated Identification
BRE	Basic Regular Expressions
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
{...}	parameters that form one structure/grouping
[x]	x is optional.
<x>	x is compulsory.
Bold	a tag to be used as is
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
xx:..xx	depicts an hexadecimal string “0” to “9” and “A” to “F”
XXX$_2$	“XXX” is binary.

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 publication of this document.

5.2 Claiming conformance

In order to claim conformance, a service shall comply with the requirements of this document.

AIDC conformance standards, as specified in Annex K, Annex L and Annex M, shall apply.

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 full test methods in Annex A shall be applied.

The test specification is independent of:

- cryptography conformance and performance;
- AutoID Data Carrier conformance and performance.

The following components shall be tested:

- DigSig Certificate format;
- DigSig Data;
- DigSig DecoderVerifier;
- DigSig DecoderGenerator.

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, read method and cryptographic functions are specified by a Domain Authority (DA) published in a DigSig Certificate (an X.509 method). The DigSig Certificate is cryptographically verifiable as certified by an X.509 Certification Authority. Each DigSig Certificate has a unique identifier (Clause 7) called the Certificate

Identifier (CID). The {DA, CID} is unique and contained in every DigSig as the first data of the DigSig. A reader of a DigSig uses the {DA, 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.

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 in order to decode and verify the data. See Annex E typical DigSig DecoderVerifier use architecture.

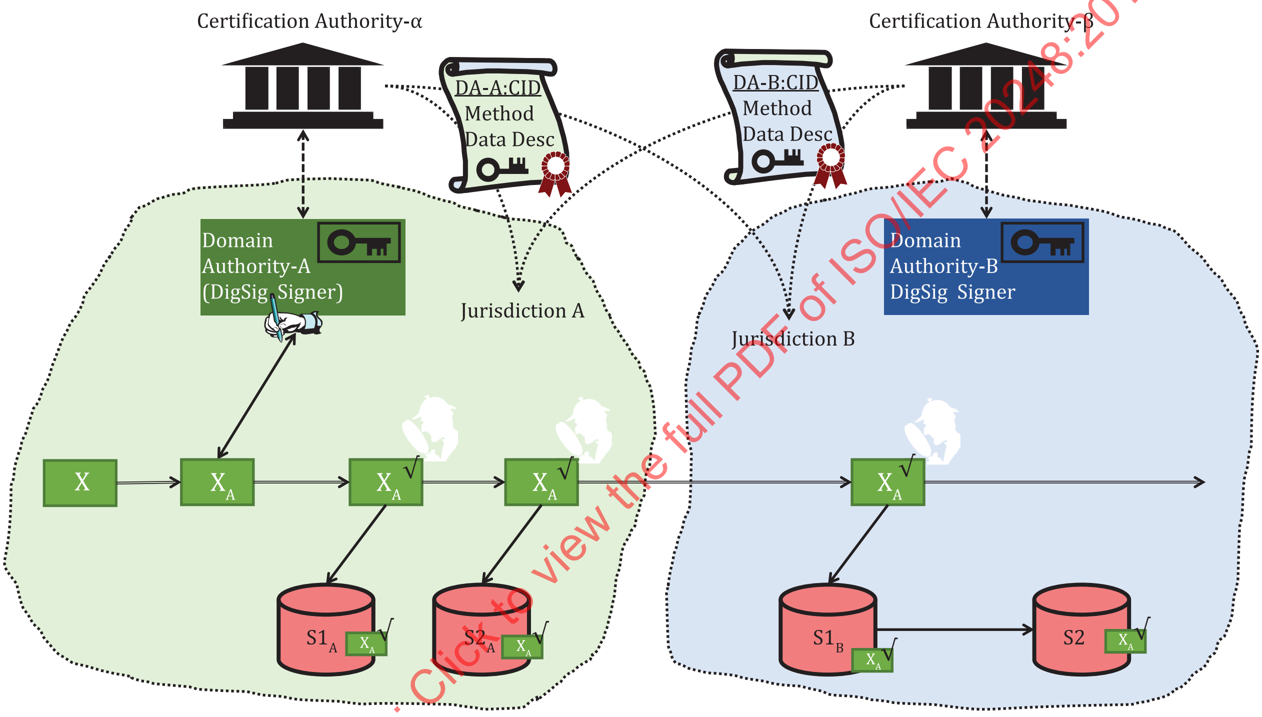


Figure 1 — DigSig use architecture

The general method and properties of ISO/IEC 20248 are illustrated in Figure 1. With reference to 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; for example:

   Data X is used by DA-A to generate a DigSigN 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.

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

Annex G provides more information on Digital Signature use.

6.2 DigSig Certificate process

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

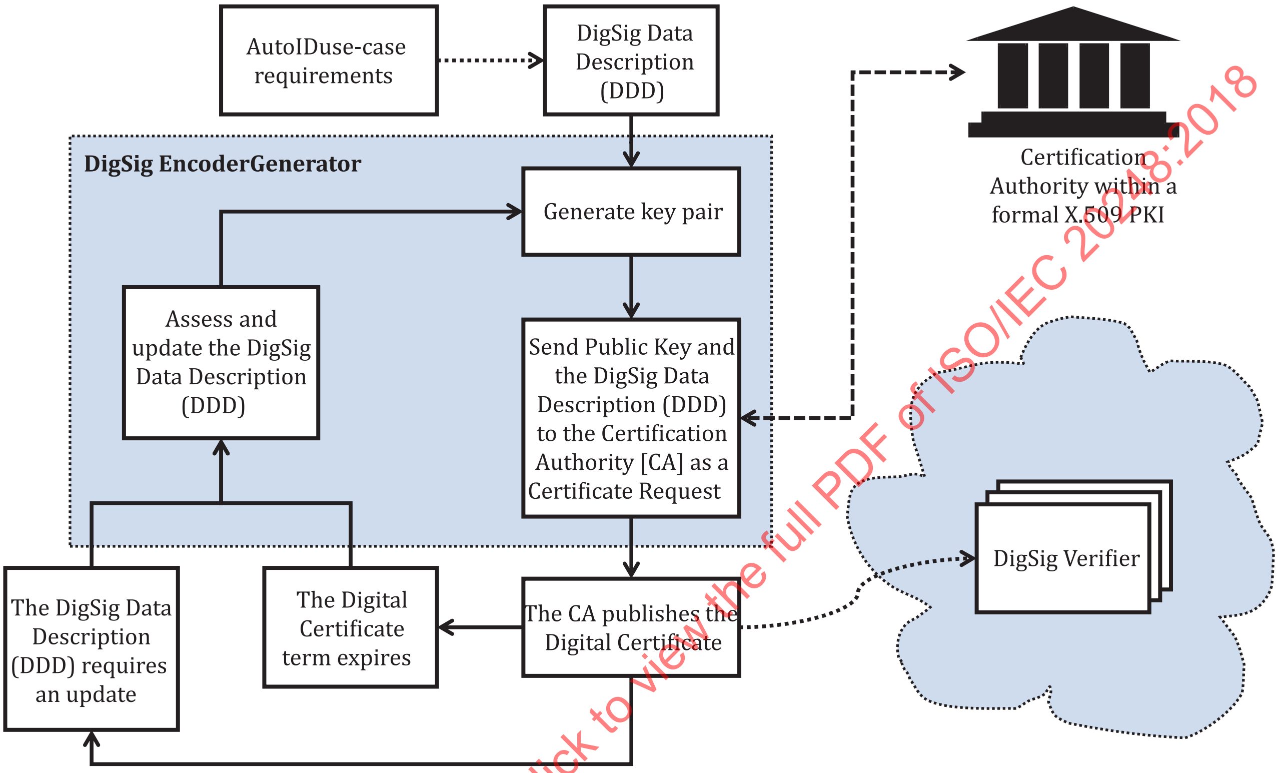

AutoIDuse-case requirements
DigSig Data Description (DDD)
Certification Authority within a formal X.509 PKI
DigSig EncoderGenerator
Generate key pair
Assess and update the DigSig Data Description (DDD)
Send Public Key and the DigSig Data Description (DDD) to the Certification Authority [CA] as a Certificate Request
DigSig Verifier
The DigSig Data Description (DDD) requires an update
The Digital Certificate term expires
The CA publishes 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.

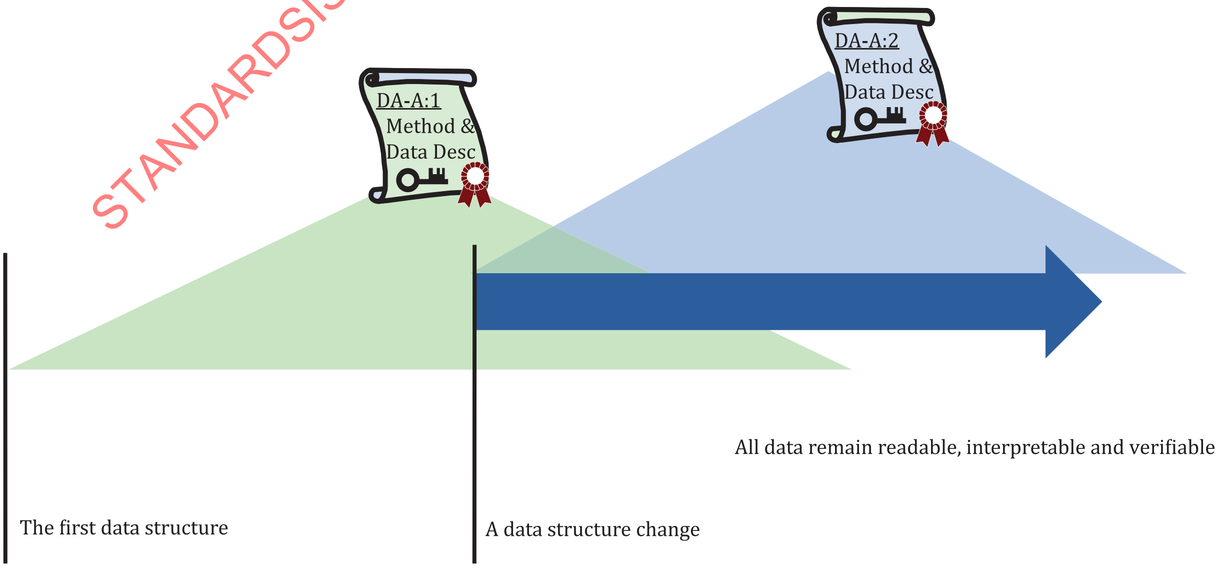


Figure 3 — DigSig data structure rollover

6.3 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 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.

6.4 DigSig verification process

The DigSig verification shall be performed by a DigSig DecoderVerifier application. The application has the ability to read 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 {DA, CID} from the DigSig Envelope. The DA is determined from the DAID. It may be derived from the URI Envelope encoding;

Use the {DA, CID} DigSig Certificate after it was verified to:

c) Read the remainder of the DigSig using the **readmethod pragma**;

d) Decode the DigSig and prepared DDDdata and SigData;

e) Perform the verification on SigData.

6.5 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 a X.509 version 3 extension.

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

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

NOTE 2 The desired CA might not be familiar with the specific requirements of this document as these differ from the standard requirements of X.509. 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 object identifier format to be used in X.509 shall be as specified in ISO/IEC 8824-1.

7.3 DigSig Certificate parameter use

The X.509 version 3 parameters (in X.509 specification notation) shall be used as follows:

```
certificateContent::= SEQUENCE {
   version                  set to the value 2 indicating certificate version 3
   serialNumber             set in accordance with X.509
   signature                set in accordance with X.509
   issuer                   set in accordance with X.509
   validity                 set in accordance with X.509 and 7.7
   subject                  {DA, CID} using distinguished names as specified in
                            ISO/IEC 9594-1 and this clause.
   subjectPublicKeyInfo     set in accordance with X.509
   issuerUniqueIdentifier   set in accordance with X.509
   Extensions::= SEQUENCE {
      extnId                ISO/IEC 20248 OID
      critical              shall have the value "non-critical"
      extnValue             DDD; see Clause 8
   }
}
```

EXAMPLE 1 See Annex H.

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

CN = <DA URI> || "/cid/" || <CID>

EXAMPLE 2 “CN = https://DomainAuthority.com/cid/12345”

NOTE 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 the Digital Signatures;
- for the 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 closely 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.

7.4.3 Private containers

The private container cryptography shall be specified by the **privatecontainer pragma** in 9.5.

7.5 DigSig Domain Authority identifier

The DigSig Domain Authority identifier shall be uniquely constructed in the following manner:

DAID |= <IAC> || " " || <CIN> (see ISO/IEC 15459-2).

An ISO/IEC 20248 Domain Authority shall be registered with a Company Identifying Number (CIN).

a) The Domain Authority identifier (DAID) shall be encoded from the ISO/IEC 15459-2 Issuing Agency Code (IAC) and CIN as specified in Table 1.

Table 1 — Domain Authority identifier bit 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 assign a unique identifier to each 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

The numerical code is converted to binary with the necessary leading zeros.

c) The rules in Table 3 shall be used to calculate a unique identifier for each CIN:

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) Numeric CIN: The numeric value is used as is.

2) Alphanumeric CIN: Base 36 conversion from the CIN to a numeric value:

i) “0” to “9” takes the character values 0 to 9.

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

iii) Leading spaces takes the character value 0.

$A_n \ldots A_2A_1A_0 \Leftarrow A_n*36^n \ldots + A_2*36^2 \ldots + A_1*36 \ldots + A_0$

NOTE ISO/IEC 15459-3 specifies that a CIN is limited to capital letters and numerals; see the Invariant Character Set of ISO/IEC 646.

EXAMPLE

15459–2 Example IAC CIN	IAC indicator	CIN indicator	IAC Code	CIN Code	DAID calculated from IAC || CIN Example
C ZZZZ	0_2	10_2	01100_2	19:A0:FF	4C:19:A0:FF
B 999999999	0_2	01_2	01011_2	3B:9A:C9:FF	2B:3B:9A:C9:FF
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

7.6 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 {DA, CID} shall be unique within its validity lifespan.

The CID values 0 and 1 are reserved for revocation, see Annex J.

7.7 DigSig validity

The Time Zone of the validity period shall be UTC.

DigSigs are issued within an issuing period. DigSigs are typically valid for a much longer period: the DigSig validity period. The validity period of the DigSig Certificate shall be at least as long as the sum of the issuing period and the DigSig validity period.

From a security and risk analysis viewpoint, X.509 allows periodic re-issuing of a new 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 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.

The revocation of a DigSig Certificate shall also revoke 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 Trust Service (see Annex E).

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

The DigSig revocation shall be as provided in 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) to a Trust Service (see Annex E).

The Trust Service shall perform the **entertext pragma** (see 9.2) through its user interface.

The Trust Service uses App services to perform the **readmethod pragmas**.

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

8 DigSig Data Description (DDD)

8.1 General

The DDD is a hierarchical 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 are called a DigSig.

AutoID Data Carriers data storage and data transfer is limited. This limitation should be considered when developing the DDD.

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

DDD |= {<digsiginfo> || <datafields>}

digsiginfo |= an object with management definition fields.

datafields |= an array of compulsory and application definition fields.

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

Figure 4 illustrates the use of a DDD.

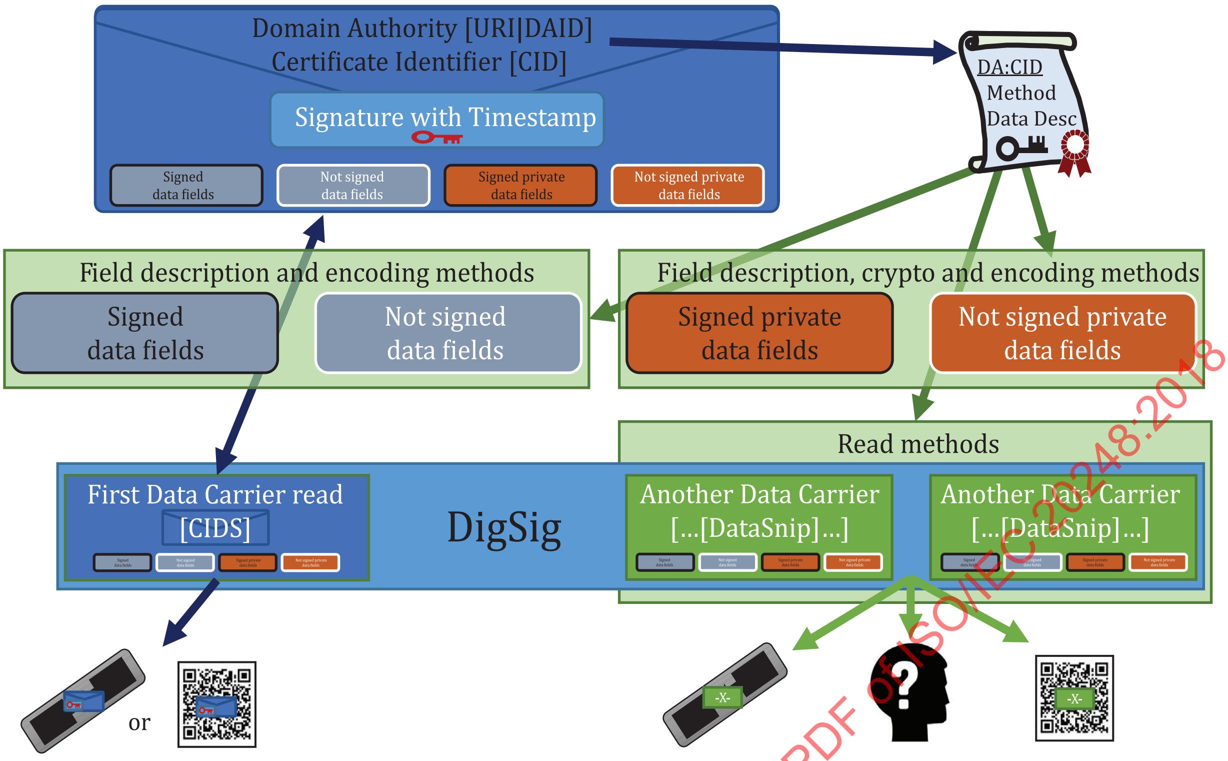


Figure 4 — Use of a DigSig Data Description (DDD)

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 For DDD, see I.3.

8.2.2 DDDdata

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

DDDdata shall be a JSON array of arrays following the structure of the DDD **datafields**.

DDDdata of a **struct** shall be contained in a JSON array.

DDDdata shall NOT contain JSON objects.

A field value shall be:

a) When cardinality is NOT specified for the field: a JSON string, number, true, false or null;

b) When cardinality is specified for the field: a JSON array of strings, numbers, trues, falses or nulls.

The values Signature and Timestamp of $\text{DDDdata}^{\text{input}}$ shall be ignored.

NOTE DDDdata are used as an input and output structure: see in Figure 4 and 8.3.4. The values Signature and Timestamp are generated by the DigSig EncoderGenerator; the input values of Signature and Timestamp are ignored.

EXAMPLE For $\text{DDDdata}^{\text{input}}$, see I.5. For DDDdata, see I.6.

8.2.3 SigData

SigData shall be 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. Empty branches shall be pruned.

SigData shall NOT contain white spaces or any other formatting.

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 for readability purposes.

DDD: {"fieldid":"id-A", "fieldid":"id-B", {"fieldid":"id-C", "cardinality":"{2}"},

{“ fieldid":"id-D","fields":{"fieldid":"id-D^1","fieldid":"id-D^2"}}}

DDDdata: [val-A, val-B, [val-C^1, val-C^2], [val-D^1, val-D^2]]

DDDdataTagged: {id-A:val-A, id-B:val-B, id-C:[val-C^1, val-C^2], id-D:{id-D^1:val-D^1, id-D^2:val-D^2}}

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.

DDDdataDisplay ⇐ GenerateDDDdataDisplay(DDD, DDDdata, Language Code, Time Zone)

DDDdata shall be generated by:

a) Adding the DDDdata field values as JSON pairs to the DDD fields in the format as specified by the DDD. The value pair shall have the format:

   “displayvalue”: <field value formatted for display - JSON string >

b) Changing all **displayStrings** to singular strings; for example:

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

Based on the selection of “af” the value of “postverify” becomes:

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

c) Changing all **date**s to the selected Time Zone and display format.

d) Purging all field members excluding: **fieldname, description** and **displayvalue**.

e) If a field member does not contain a **fieldname** pair, the **fieldid** shall be used for display purposes.

EXAMPLE See I.9.

8.3 DigSig format

8.3.1 General

See Figure 4. The encoded form of the DDD data is called a DigSig. A DigSig may be stored over more than one Data Carrier. It contains fields which are signed or not signed and open or private.

The DigSig Envelope is the encapsulation of the {DA, CID} and the data fields as specified to be stored on the Data Carrier containing the {DA, CID}.encoded format of the DDDdata. The DigSig Envelope is the first part of the DigSig read.

{DA, CID} references the DigSig Certificate, which contains the DDD, which in turn defines methods to read, decode and verify data of the DigSig.

NOTE Human memory is also a Data Carrier. Data are provided from human memory as a TEXT Snip.

8.3.2 Snips

8.3.2.1 Snip types

A Snip is a singular continues part of a Data Carrier memory as specified by the DDD. It contains one or more fields.

The following Snips shall be used:

a) CIDSnip: The CIDSnip is the head, the first Snip, of the DigSig. It shall contain the DAID, CID and all data fields without a **readmethod pragma**.

b) DataSnips: The additional Snips as specified by the DDD **readmethod pragma**. Each **readmethod** shall map to a DataSnip.

The DataSnips shall be in the order of reference in the DDD.

NOTE Data Carrier typically will provide a CIDSnip in an automated manner; for example 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 CIDSnip shall be encoded in BINARY. A DataSnip shall be encoded as either BINARY or TEXT:

a) BINARY encoding shall be packed as a bit stream with the most significant bit first in network byte order.

b) TEXT encoding shall be UTF.

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.

EXAMPLE See I.4.

8.3.3 Envelope format

8.3.3.1 General

The Envelope specification contains fields specified in 8.5.

8.3.3.2 RAW Envelope

A Raw Envelope is the CIDSnip.

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 URI Envelope shall have the following format:

URI Envelope |= <verificationuri> || “?” || Base64url_encoding(CIDSnip) [|| “~” <cidsniptextvalue>]…

The **cidsniptextvalue** shall be in the order of reference in the DDD; see 9.7.

NOTE The URI Envelope is useful for online verification where a Data Carrier is read using a hand-held device with generic Data Carrier readers; e.g. a smart phone with a barcode and/or RFID reader.

8.3.4 AIDC specific construction of a DigSig

Each Snip of a DigSig shall be specifically constructed for insertion into the desired Data Carrier allowing the DigSig to be read correctly from such Data Carrier.

This international standard defines a process, as depicted in Figure 5, for encoding a DigSig in data carriers. Rules are explicitly specified for:

- 2 D barcodes (see Annex K for details), specifically:
  - Aztec Code, as defined in ISO/IEC 24778;
  - DataMatrix, as defined in ISO/IEC 16022;
  - MicroPDF417, as defined in ISO/IEC 24728;
  - PDF417, as defined in ISO/IEC 15438;
  - QR Code, as defined in ISO/IEC 18004.
- RFID tags to these protocols:
  - ISO/IEC 18000-3 Mode 1, operating at 13.56MHz (see Annex L for details);
  - ISO/IEC 18000-63, operating at 860 MHz to 960 MHz (see Annex M for details).

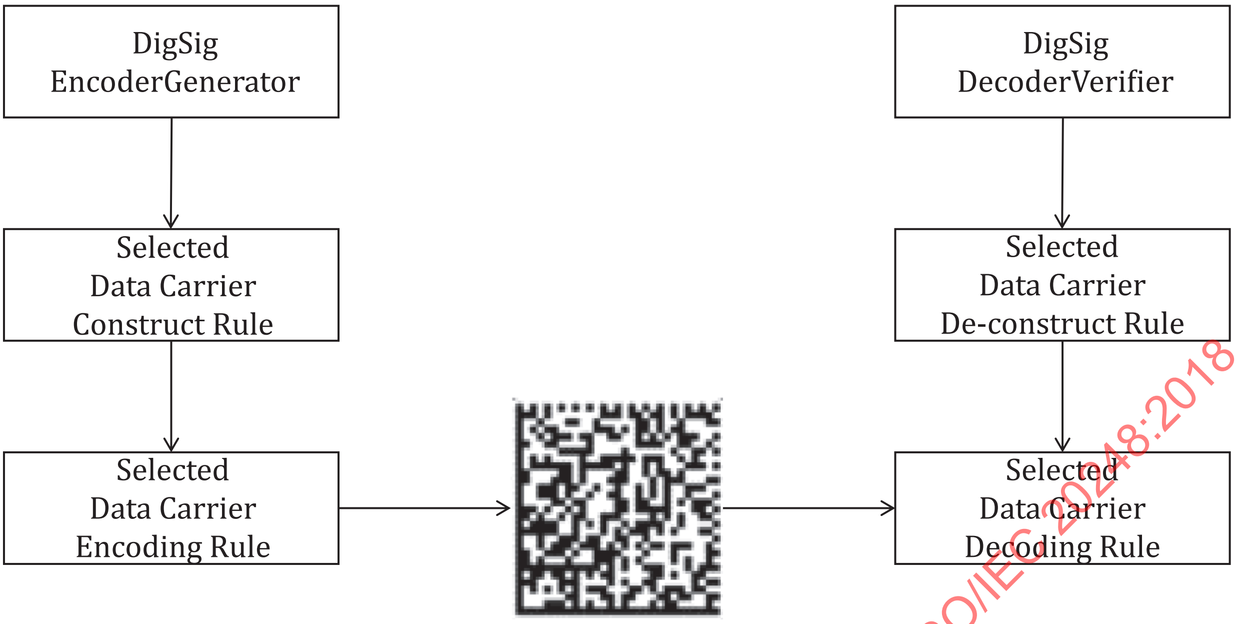


Figure 5 — Encoding and decoding a DigSig in AIDC data carriers

The data carrier construct rule addresses the following features.

- The data carriers need to declare that the encoding complies with the International Standard. There are different ways of doing this in RFID and bar code data carrier.
- The Domain Authority and CID need to be processed in a manner specific to the data carrier. For example, these are encoded in different memory areas in the different RFID protocols, while all the 2D symbols share common features.
- RFID tags support selective locking.
- ISO/IEC 18000-63 supports addressable partitioned memory and these memory areas may require the DigSig to be distributed between these memory areas.
- The 2D symbologies support error correction, sometimes with a choice of correction level.
- The 2D symbologies require print parameters to be defined, including defining the format of the symbol.

A similar inverse process is required to de-construct the data carrier. Additionally, the DigSig DecoderVerifier may call for additional processes to be carried out based on the content of the data carrier, e.g. to use the unique chip id in an RFID tag.

8.4 The DigSig physical data path

The physical data path of a DigSig is depicted in Figure 6.

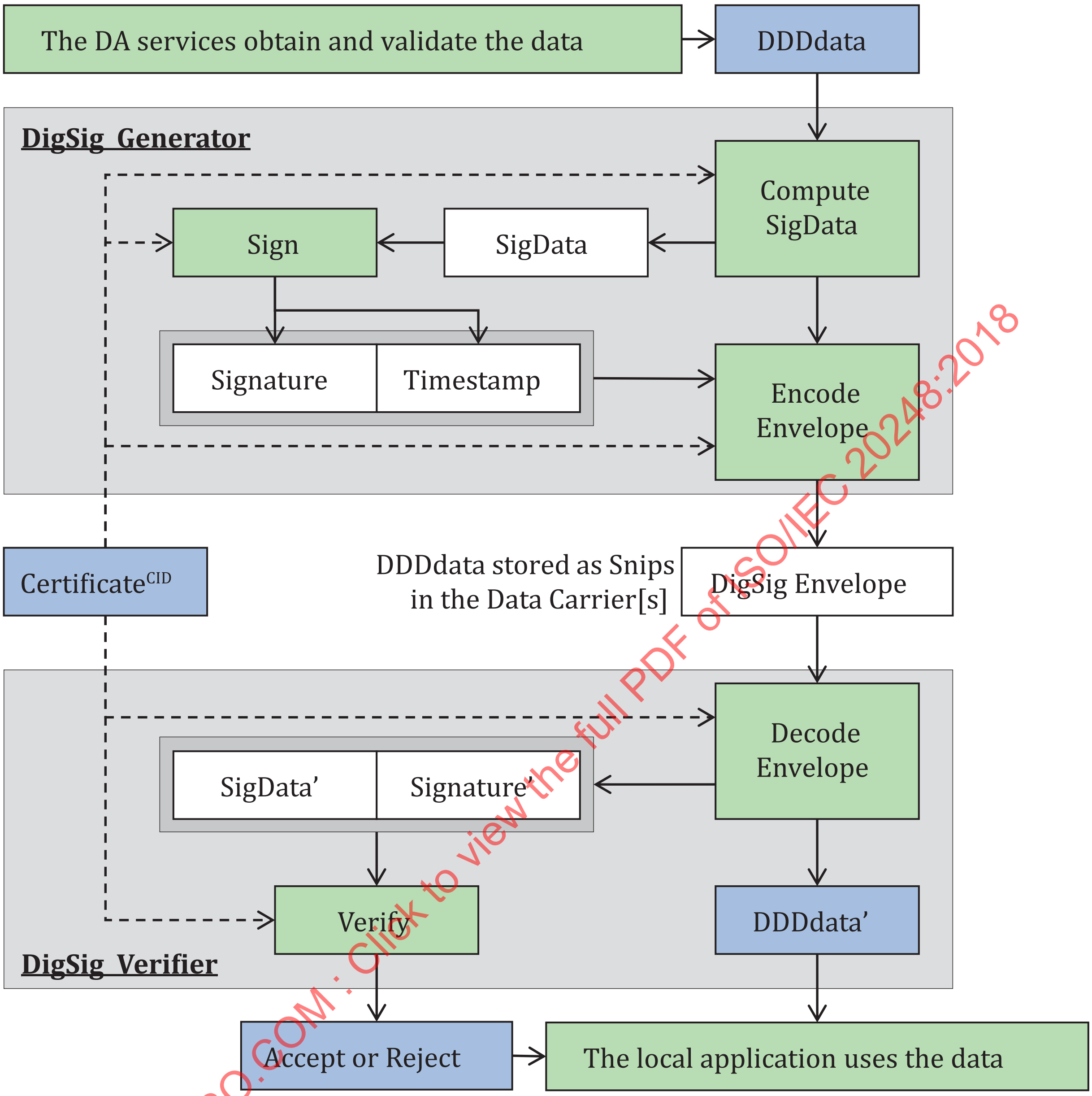


Figure 6 — Physical data path

a) DDDdata^{input} ⇐ LocalApplicationGenerateDDDdata(Certificate^{CID}, ApplicationData)

b) SigData ⇐ ExtractSigData(Certificate^{CID}, DDDdata^{input})

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

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

e) Programmed Data Carrier… ⇐ WriteDigSig(DigSig)

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

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

h) VerificationOutcome ⇐ Verify(Certificate$^{CID'}$, SigData’, Signature'}

i) ApplicationData ⇐ TranslateFromLocalApplication(Certificate$^{CID'}$, DDDdata', VerificationOutcome)

8.5 DDD syntax

The DDD shall have the following syntax:

```
{"digsiginfo":
  {"specificationversion":<specificationversionValue>,
   "dauri":<dauri>,
   "daid":<DAID>,
   "cid":<CID>,
   "verificationuri":<verification URI>,
   "revocationuri":<revocation URI>,
   "preverify":<display string before verification>,
   "acceptverify":<display string on verification accept>,
   "rejectverify":<display string on verification reject>,
   "postverify":<display string after verification>,
   "structureddocuri":<structured document URI>,
   "custom":<JSON member with custom DDD information>
   },
  "datafields":
   [{"fieldid":"specificationversion",    # Shall be the first field description
     "type":"string",
     "bsign":true
    },
    {"fieldid":"dauri",                   # Shall be the second field description
     "type":"string",
     "bsign":true
    },
    {"fieldid":"daid",                    # Shall be the third field description
     "bsign":true
    },
    {"fieldid":"cid",                     # Shall be the fourth field description
     "type":"unsignedint",
     "bsign":true
    },
    {"fieldid":"signature",               # This field may be placed elsewhere
     "type":"bstring",                    # Fixed length as per DigSig Certificate
     "bsign":false
    },
    {"fieldid":"timestamp",               # This field may be placed elsewhere
     "type":"date",                       # The Time Zone shall be UTC
     "binaryformat":<dateformat>,         # The default is tepoch
     "bsign":true
    },
    …                                     # Optional application fields
   ]
}
```

specificationversion, **dauri**, **cid**, **signature** and **timestamp** are compulsory **datafields**. The compulsory fields shall only use the field descriptors with its specified value as specified in this clause. The field descriptors may be omitted from the DDD.

8.6 DigSig information fields

The DigSig information fields describe the use environment of the DigSig and DDD.

digsiginfo is a JSON object. It shall contain only the fields specified in Table 4. These fields shall not contain DDDdata values.

Table 4 — digsiginfo fields

Name	Purpose and value	Type	Inclusion
specificationversion	Set according to 5.1.	JSON string	compulsory
dauri	The Domain Authority URI for reference purposes.	JSON string	compulsory
daid	The Domain Authority identifier.	JSON string	compulsory

Table 4 *(continued)*

Name	Purpose and value	Type	Inclusion
cid	The DigSig Certificate identifier, see 7.6.	unsigned integer	compulsory
verificationuri	This URI points to an online verification method. It is also used in the URI Envelope. It is recommended to be the same as dauri.	JSON string	compulsory
revocationuri	This URI points to the revocation method.	JSON string	compulsory
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
structureddocuri	This URI shall point to a structured document which uses the fields.	JSON string	optional
custom	JSON object with DDD custom information.	JSON object	optional

8.7 Data fields

The DDD data fields specify the information, structure, read method and encoding of the DDDdata fields. DDD data fields shall not have values.

datafields is on JSON array of objects.

datafields shall contain the compulsory data fields as depicted in Table 5.

8.7.1 Compulsory data fields

The compulsory fields describe the DigSig data fields.

Table 5 — 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	Yes	Third	First
cid	16 bit unsignedint	Yes	Fourth	Second
signature	fixed length **bstring**	No	Default fifth	Default third
timestamp	date with binaryformat	Yes	Default sixth	Default fourth

The following special rules shall apply.

a) Compulsory data fields shall only contain the **type** and **bsign** field descriptors which shall be set according to Table 5. **timestamp** may optionally contain **binaryformat**.

b) **signature** shall be generated at the point of signing using the DigSig Certificate specified signature algorithm and its specified bit length.

c) **timestamp** shall be generated at the point of signing using the specified resolution in accordance with the binaryformat setting. The Time Zone shall be UTC.

8.7.2 Application data fields

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

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 shall contain a set of field descriptors as specified in Table 6.

Table 6 — 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
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.	null	displaystring	optional
description	Display string intended for display purposes on capturing, printing or extracting the data.	null	displaystring	optional
type	Defines the DDD field type as bstring, string, unsignedint, number, date, enum, boolean or struct.	—	JSON string	compulsory
	The compulsory fields specified in Table 5 takes the type as specified in Table 5.	As specified in Table 5	—	not included
decimalunit	Specifies the accuracy of a **number**.	0	integer	optional with the type **number**
truncation	Specifies the truncation of the input value of a **number**.	“round”	string	optional with the type **number**
bsign	<**true** | **false**>; if true then the field value is included in the signature.	true	boolean	optional
nullable	<**true** | **false**>; if true then the field value may be set to null.	false	boolean	optional
nulldescription	Display string to describe a **nullable** field when it contains a **null**.	null	displaystring	optional
nonnulldescription	Display string to describe a **nullable** field when it contains a value.	null	displaystring	optional
cardinality	Repeat the field in a JSON array.	—	JSON string	optional
range	The allowed value range of the field.	—	As specified by **type**	optional
binaryformat	This is a field member specifying the binary format. Binary format is associated with **type**.	—	JSON string	optional
pragma	See Clause 9.	—	JSON object	optional
fields	**value** is a JSON object that contains the fields of the type **struct**.	—	—	used with the type **struct**

Table 6 *(continued)*

Name	Purpose and value	Default	Type	Inclusion
displayformat	Formatting to be applied to data when it is displayed. Standard C formatting (in accordance with ISO/IEC 9899) shall be used. Date formats shall use strftime representing date as a string. The Zone "Z" shall be displayed as specified in 8.9.3.7 using **timezone**.	According to **type**. For **date**: "YYYY-MM-DDThh:mmZ"	JSON string	optional
timezone	See 8.9.3.7.	"UTC"	JSON string	optional
defaultvalue	The default value for the field when capturing the data.	—	As specified by **type**	optional
custom	JSON object with DDD custom information. Components of this standard shall ignore this field.	—	JSON object	optional
displayvalue	The field value in display format. **displayvalue** shall only be used in DDDdataDisplay.	—	JSON string	—

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 DDD types which map onto the JSON types to include: **unsignedin**t, **number**, **bstring** and **date**. The JSON type **for unsignedint**, **number** 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;

— struct maps to a JSON object containing the fields of the struct.

8.9.2 Special field values

8.9.2.1 The DDD field value "null"

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

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 a null-bit in the MSB position 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.

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

If **nullable** is set to **true**; then the value of **nulldescription** or **nonnulldescription** shall be displayed.

NOTE See 9.4 for the **null** bit encoding where the **readmethod pragma** is specified.

8.9.2.2 Array of field values - cardinality

cardinality specifies a repeat of the field. The repeated field values shall be an array of values in DDDdata.

The value of **cardinality** shall be the JSON string "variable" which specifies a variable **cardinality** or for a specified cardinality a POSIX ERE "{}" bracket expression (in accordance with ISO/IEC IEEE 9945) encoded as follows:

m > 0, 0 < n ≤ m

{m} specifies a fixed number of repeats. The binary encoding shall reserve m fields.

{n,m} specifies variable repeats of n to m. The binary encoding of a variable cardinality field shall be prefixed with count-bits. The count-bits shall be encoded as follows:

count-bits |= {<countsize> || <fields>}

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

fields is the number of parts encoded as a multiple of nibbles ($CCCC_2$).

EXAMPLE 1 count-bits 0_2 specifies no field values.

EXAMPLE 2 count-bits 10_2 $CCCC_2$ specifies 0 to 15 field values.

EXAMPLE 3 count-bits 110_2 $CCCC_2$ $CCCC_2$ specifies 0 to 255 field values.

The DDDdata value shall be a JSON array of values.

8.9.3 Field types

8.9.3.1 boolean

The binary encoding of a **boolean** field is 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 field is 2 bits long, for example: false: 10_2; true: 11_2; null: 0_2.

binaryformat and **range** have no meaning and shall be ignored.

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

8.9.3.2 unsignedint

An **unsignedint** field shall contain an integer with range zero to 2^{32}-1.

binaryformat may be used to set the maximum range value using a bit length encoding. The value of **binaryformat** shall be an integer b. The binary encoding shall allocate b bits for the field value.

The maximum range value shall be 2^b-1 with 0 < b ≤ 32.

The default value of b is 16.

range may be used to define an inclusive field value range n to m with 0 ≤ n ≤ m ≤ 2^{32}-1. **range** shall be specified with the range block notation "[n..m]". The binary encoding shall allocate RoundUp(LOG$_2$(m-n)) bits for the relative field value using n as the base. **binaryformat** shall be ignored if **range** is specified.

The DDDdata value shall be a JSON number greater or equal to zero.

8.9.3.3 number

A **number** field shall contain a number accurate to a decimal unit $10^{decimalunit}$.

EXAMPLE 1 If **decimalunit** = 2, the numbers are multiples of 100s; for example 12300, 100, −100, 4560000, etc.

EXAMPLE 2 If **decimalunit** = −2, the numbers are multiples of 100ths; for example 1042.34, −0.25, 0.00, −13242.12, etc.

The **number** value shall be represented as follows:

number value = <encoded integer> * $10^{decimalunit}$

The encoded integer default range is -2^{b-1} to 2^{b-1}-1 with $0 < b \le 32$.

The value of **decimalunit** shall be an integer. The default is zero.

binaryformat may be used to set the minimum and maximum range values of the encoded integer using a bit length encoding. The value of **binaryformat** shall be an integer b. The binary encoding shall allocate b bits for the encoded integer value.

The range value shall be (-2^{b-1} to 2^{b-1}-1) * $10^{decimalunit}$ with $0 < b \le 32$.

The default value of b is 16.

range may be used to define an inclusive field value range n to m with ($-2^{31} \times 10^{decimalunit}$) ≤ n, *n* ≤ m and m ≤ 2^{31}-1 × $10^{decimalunit}$). **range** shall be specified with the range block notation “[n..m]”. The binary encoding shall allocate RoundUp(LOG$_2$((m-n)/$10^{decimalunit}$)) bits for the relative field value using n as the base. **binaryformat** shall be ignored if **range** is specified.

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

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

round |= round to nearest with tie breaking towards **ceiling**;

floor |= round down; truncate the fraction

round is the default.

The DDDdata value shall be a JSON number.

8.9.3.4 string

A **string** field contains a UTF-64 printable string encoded as an enumeration.

A **string** is case sensitive.

binaryformat specifies the number of characters of the string. If **binaryforma**t is not specified, the **string** shall be variable length.

The value of **binaryformat** shall be a JSON string using the POSIX ERE “{}” bracket expression as specified in ISO/IEC IEEE 9945. The string shall be encoded as follows:

0 < *n* ≤ m

{m} specifies a fixed length **string** of m characters. The string shall be padded at the end with the first character of the range. This character has a binary encoding of zero.

{n,m} specifies a variable length string of n to m characters. The binary encoding of a variable length string shall be prefixed with length-bits. The length-bits shall be encoded as follows:

length-bits |= {<lengthsize> || <num_chars>}

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

num_chars is the number of characters represented in **lengthsize** nibbles; $CCCC_2$

EXAMPLE 1 length-bits 0_2 specifies an empty **string.**

EXAMPLE 2 length-bits 10_2 $CCCC_2$ specifies a string with 0 to 15 characters.

EXAMPLE 3 length-bits 110_2 CCCC2 $CCCC_2$ specifies a string with 0 to 255 characters.

Range may be used to limit the allowed characters within the **string** with a single "[]"bracket BRE expression. The bit encoding of each character shall be in the BRE "[]" sequence starting with zero. The encoding shall be as specified in Table 7.

The default range is [:print:] or [\x20-\x7E]; the POSIX visible characters and the space character.

Table 7 — Bit encoding ranges

Range size	Bits per character
1 to 2	1
3 to 4	2
5 to 8	3
9 to 16	4
17 to 32	5
33 to 64	6
65 to 128	7
129 to 255	8
...	...

EXAMPLE In an application that requires 6 characters to be limited to 0, 1, 2, 3, A and B, the POSIX ERE "{}" 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.

The DDDdata value shall be a JSON string.

8.9.3.5 bstring

A **bstring** field contains m bits of binary data. The DDDdata value shall be the Base64URI encoded value of the **bstring**.

binaryformat specifies the number of bits in the **bstring.** If **binaryformat** is not specified, the **bstring** shall be variable length.

The value of **binaryformat** shall be a JSON string using the POSIX ERE "{}" bracket expression as specified in ISO/IEC IEEE 9945. The string shall be encoded as follows:

m > 0, 0 < n ≤ m

{m} specifies a fixed number of bits in **bstring.** The binary encoding shall reserve m bits.

{n,m} specifies a variable length **bstring** of n to m bits. The binary encoding of a variable length **bstring** shall be prefixed with length-bits. The length-bits shall be encoded as follows:

length-bits |= {<lengthsize> || <num_bits>}

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

num_bits is the number of bits in **lengthsize** nibbles; CCCC2…

EXAMPLE 1 length-bits 0_2 specifies an empty **bstring**.

EXAMPLE 2 length-bits 10_2 $CCCC_2$ specifies a string with 0 to 15 bits.

EXAMPLE 3 length-bits 110_2 $CCCC_2$ $CCCC_2$ specifies a string with 0 to 255 bits.

The DDDdata value shall be a JSON string, with the bit string encoded in Base64url.

range have no meaning and shall be ignored.

8.9.3.6 digsigenv

The **digsigenv** is a special designated **bstring** indicating that it is a nested DigSig Envelope.

All the **bstring** rules apply to **digsigenv**.

The SigData of **digsigenv** shall be the encoded bit string of the nested DigSig Envelope.

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

8.9.3.7 date

A **date** field contains an exact 32 bit POSIX epoch date as specified in ISO/IEC IEEE 9945 or a date relative to **CertificateContent.validity.not before** of the DigSig Certificate.

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

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** is specified using **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 TimeZone and append the device Time Zone.

tag |= The DDDdata shall be Time Zone agnostic. A relative date shall use **CertificateContent.validity.not before** in a Time Zone agnostic manner. DDDdataDisplay shall NOT append a Time Zone.

NOTE DDDdataDisplay is generated by the DecoderVerifier.

range shall be defined with the range block notation “[n..m]”, specifying an inclusive date range n to m. n and m are dates in the tepoch display format.

The DDDdata value shall be a JSON number.

8.9.3.8 enum

An **enum** is a single selection of a 0 based index of an enumeration of values. The syntax of an **enum** field shall be as follows:

```
{
"fieldid":<field short name>,
"type":"enum",
"enumvalues":<enum values>,
"enumvaluedesc":<enum display values>,
"bsign":<signature inclusion>,
<optional field descriptors>,
}
```

The **enum** descriptors is specified in Table 8.

Table 8 — 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

binaryformat, **range** and **defaultvalue** shall be ignored.

The binary encoding of an **enum** is an **unsignedint** of b bits. b shall be calculated as specified in Table 9.

Table 9 — enum binary encoding

enum size	b
1 to 2	1
3 to 4	2
5 to 8	3
9 to 16	4
17 to 32	5
33 to 64	6
65 to 128	7
129 to 255	8
...	...

The DDDdata value shall be a JSON string selected from enumvalues.

8.9.3.9 struct

The **struct** type shall be used to group fields in a hierarchical manner. The fields of a **struct** shall be an array of fields with the field member name: **fields**. The syntax of a **struct** field shall be as follows:

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

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

displayformat, **binaryformat**, **range** and **defaultvalue** shall be ignored.

NOTE 1 **cardinality** repeats the **struct**.

NOTE 2 a **struct** is nullable.

The DDDdata value shall be a JSON array of fields.

8.9.4 Special types

8.9.4.1 General

The special types shall not define nor specify DDDdata. It is used for supplementary functions.

8.9.4.2 displaystring

displaystring provides language translation for display strings. Display string shall not be part of DDDdata. It is used to describe the capture and display of DDDdata.

displayString is a JSON value object with at least one element which specifies the language and the text as a string. The first element is the default. The syntax is as follows:

{<Language Code>:<string to be displayed>[, ...]}

Language Code shall be constructed in accordance with RFC 5646. The use of ISO 639-1, 2 character language code is compulsory. The RFC 5646 subtags are optional.

EXAMPLE 1 displaystring: {”en”:“This is the display text”, “af”:“Hierdie is vertoonteks”, “de-AT”:“Dies ist der Anzeigetext “}

EXAMPLE 2 Language Codes: below a list of language variation examples.

English: “en”:“January”, Austrian German: “de-AT”:“Jänner”, German: “de”:“Januar”, Swiss German “de-CH”:“Januar”, Swiss French “fr-CH”:“Janvier”

9 Pragmas

9.1 General

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

Pragmas shall NOT be nested.

The syntax of **pragma** shall be as follows:

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

9.2 entertext

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

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

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

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

entertext may be used with **structjoin**.

The input format of for a field type shall be as follows.

— **boolean**: “TRUE" or "FALSE” in a selection list.

— **unsignedint**: Input starting with "0x" shall be hex. All other input shall be a positive decimal integer.

Decimal: 9..99 is digital with “9” the characters “0” to “9”.

Hex: 0xX..XX is hex with “X” the characters “0” to “9” and “A” to “F” not case sensitive.

— **number**: Decimal number with the characters minus, space, point and "0" to "9". The input may indicate the decimal unit.

— **string**: The valid characters as specified with **range**.

— **bstring** and **digsigenv**: White spaces are allowed and will be stripped. The following formats shall be supported:

Binary: b..bb is binary with “b” the characters “0” and “1”

Hex: XX:..XX is hex binary with “X” the characters “0” to “9” and “A” to “F” not case sensitive. Hex binary is indicated with the use of “”: in the input string. One or more “”: may be placed anywhere in the input string.

Base64url: consisting of the Base64url characters.

— **date**: as specified with **range**.

— **enum**: The enumeration is selected from a selection list.

9.3 structjoin

structjoin is used to join a DDD structure into a single DDD **string**.

structjoin shall only be used with **struct**.

DDDdata and SigData shall contain the **struct** NOT joined. DDDdataTagged and DDDdataDisplay shall contain the joined structure as a singular string.

The **struct** fields shall be joined using standard C formatting.

<structjoin format> shall be a JSON string and contain standard C formatting (in accordance with ISO/IEC 9899) for the fields of the structure.

EXAMPLE In the following field description, the field **licencenumber** in SigData contains a value “LN 345ABC” for which the field **LN-999part** contains “345” and the field **LN-AAApart** contains “ABC”.

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

9.4 readmethod

readmethod is used to specify the read method of a field. Specifying a **readmethod** shall result in one or more additional DataSnips. The read method is Data Carrier specific.

Table 10 — 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.	—	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
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

The syntax of **readmethod** shall be as follows:

```
"readmethod":
 {"methodname":<method name>,
  "methoddesc":<an operator instruction>,
  "methodmemory":<memory segment location>,
  "methodsnipencoding":<"BINARY" | "TEXT">,
  "methodoffset":<JSON location array>,
  "methodlength":<length to be read>
 }
```

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

The **readmethod** descriptors are specified in Table 10.

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":"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 a **struct** which is encrypted. The **struct** may be stored in an encrypted format, or received over the air in an encrypted format.

privatecontainer shall only be used with **struct**.

DDDdata and SigData shall contain the **privatecontainer** as a singular **bstring**.

DDDdataTagged and DDDdataDisplay may contain the **privatecontainer** as the specified decoded fields.

The local application shall decrypt the private container where after the DDD decoding of the **privatecontainer** fields can take place.

It is recommended that the scope of a decoded private container is limited to the local 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 11.

Table 11 — privatecontainer field descriptors

Name	Purpose and value	Default	Type	Inclusion
privatecontaineruri	The URI where the local application will find the methods and keys to decrypt a private container. This URI should be security protected. The local application should be authenticated.	—	JSON string	compulsory
privatecontainerdesc	A display string to optionally instruct an operator; e.g. where a Data Carrier is located.	—	displaystring	optional

9.6 startonword

startonword is used to align a field value bit 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**.

9.7 cidsniptext

cidsniptext is used to encode a **string** field as TEXT in the URI Envelope.

NOTE This is useful where fields needs to be human readable in a URI Envelope. It is typically used with barcodes where only a CIDSnip is specified.

cidsniptext shall only apply

— to **string** fields on the first layer of the DDD (i.e. the field is NOT in a structure), and

— to the URI Envelope.

The syntax of the **cidsniptext** value shall be as follows:

cidsniptextvalue |= /[headtext]<field value>[tailtext]

The field value, the **headtext** and **tailtext** exclude the character "~" and are limited to the non-reserved URI character set as specified by RFC 3986. The reserved characters within the URI character set are:

: / ? # [ ] @ ! $ & ' ( ) * + , ; =

The syntax of **cidsniptext** shall be as follows:

```
"cidsniptext":
  {"headtext":<Non reserved URI characters head text>,
   "tailtext":<Non reserved URI characters tail text>
  }
```

The **cidsniptext** descriptors are specified in Table 12.

Table 12 — cidsniptext field descriptors

Name	Purpose and Value	Default	Type	Inclusion
headtext	Non reserved URI characters inserted at the start of the field value.	“”	JSON string	optional
tailtext	Non reserved URI characters inserted at the end of the field value.	“”	JSON string	optional

The cidsniptextvalue shall be appended to the URI Envelope as specified in 8.3.3.3.

EXAMPLE https://www.dept-edu.com/verifyme?ekjcsafg6IOVbnNHSe7FMWqtMQNAtuwo3bgY~25S-ABC123~26S-ABC456~27S-FGH987

25S-ABC123, 26S-ABC456 and 27S-FGH987 are **cidsniptext** with "25S-", "26S-" and "27S-" **headtext.**

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 specified in ISO/IEC 9594-8 and this document.

A.1.3 Test method

The test method shall be as follows:

- 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;
- 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

In order for the DigSig DecoderVerifier to be tested, the following shall be provided.

— For test by norm application: the executable loaded on the intended platform with all the required peripherals.

— For 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 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

In order 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;
- 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 attempts 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.

a) 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.

b) The data read are unaltered from when they were written.

c) The data were authorised at the moment 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

~~**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 therefore need not be shown in the shorthand. The **daid** and **cid** shall be the first fields. The ~~**signature**~~ and **timestamp** fields, by default, follow the **daid** and **cid** fields 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, SecretPIN}

EXAMPLE 2 ItemDigSigNFC{ItemID, OwnerID, SecretPIN}

Examples 1 and 2 represent the same DigSig.

EXAMPLE 3 ItemDigSigNFC{daid, cid, ItemID, OwnerID, SecretPIN, signature, timestamp}

The signature and timestamp are explicitly placed at the end of the DigSig.

EXAMPLE 4 ItemDigSigNFC{ItemID, OwnerID, SecretPIN, signature, timestamp}

The symbols "[]" indicate an array of values for a specific field.

EXAMPLE 5 "field1" indicates field1 has one value. "field1[]" indicates field1 has one or more values.

The symbols "||", "…" and [structure name]{…} may be used to describe more complex DigSigs in shorthand.

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

To aid readability, fields may 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{}}

List{{ItemID, ItemDesc}…} stored in the QR code on the waybill.

B.3 A simple timestamp

A simple timestamp can be created with an empty DigSig:

Timestamp{}

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 would 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.4 Barcode DigSigs for mobile phones

Barcodes to be read by mobile phones are commonly generated in TEXT. It will generally contain a URI which tells the mobile phone what to do with it. The associated data are often also encoded in TEXT or contained in a sparse proprietary data structure. The result is often a very large barcode, which reduces the read quality of the barcode. A dedicated app and a real time connection are often also required. This document provides an automated compaction of the data which can be decoded, interpreted and verified without the need for a real time connection to a central service.

It is also easy to tamper with a barcode simply by reading a barcode, changing parts of it and recreating it. This may lead to a barcode which points to a rogue URI. This document allows a mobile phone (and its user) to detect such tampering. An example of a DigSig barcode is provided in Annex I.

A DigSig however, does not provide a method to detect copying of the 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 the **readmethod**; IDcardLinearBarcode 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** field with a **readmethod pragma** with **methodname** ["ISO/IEC 15417"] and OwnerMemory{PIN} is a **struct** field with a **entertext pragma**.

The barcode is fixed on the computer. When the barcode is read the DDD will

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

b) Instruct the standard DigSig DecoderVerifier App on the phone to display the ComputerSerialNumber for inspection,

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

d) Request 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.5 A RFID example

The issue of originality can be addressed when an electronic Data Carrier is used. An English language competency certificate example may be constructed as follows:

IELTS{{ACADEMIC | STANDARD}, Date, Centre,
 Candidate{Number, ID, FamilyName, Names[], Nationality$^{\text{ENUM}}$, FirstLanguage$^{\text{ENUM}}$}
 Results{Listening, Reading, Writing, Speaking, BandScore, CEFR}, TID{TagID$^{\text{Nullable}}$}}

NOTE TID{TagIDNullable} is a nullable **struct** field with a **readmethod pragma**, with **methodname** [“ISO/IEC 18000-63”], with **methodmemory** 2, **methodbitoffset** 0 and **methodbitlength** 96.

With this DDD we have two applications, which can either be used independently or together to facilitate the best of both approaches.

a) Barcode DigSig – the TagID shall be set to **null**. This DigSig barcode is applied on the certificate. This allows the verification of the information of a copied certificate.

b) RFID DigSig – the TagID shall be the TagID of the RFID tag. The RFID tag is also applied to the original certificate. If the TagID is unique then the originality can also be verified by reading the unique TagID.

NOTE Typically the TagID of the Data Carrier is stored in a different memory segment from the memory segment which is read in the first Data Carrier interrogation. The DDD needs to specify a **readmethod** for TagID which will result in an additional DataSnip following the CIDSnip read in the first interrogation.

B.6 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 be 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.

— The DigSigRecords should be written from the end of the memory.

B.7 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.6. It should be noted that, in this complete offline example, each role player is deployed with a DigSig EncoderGenerator which falls under the public key infrastructure of a domain authority.

Let us consider the lifecycle of a vehicle licence plate, which is similar to 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.

a) 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.

b) An Embosser will add the licence number to the plate after the Embosser assured itself that the blank is genuine and the licence number is legal. These plates are sent to the issuing point.

c) 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.

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 takes action 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{CID, LicenceNumber$^{\text{Nullable}}$, VehicleVisuals{}$^{\text{Nullable}}$, LicenceExpiry},
TID{TagID},
UserMem{~~Signature~~, TimeStamp, BlankSerial, ~~BlankDigSig{}~~$^{\text{Nullable}}$,
~~EmbossDigSig{}~~$^{\text{Nullable}}$}}

NOTE The position of the fields optimises both the reading and writing of the Data Carrier in the various use cases.

The actions 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.

NOTE 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.

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.

This can seem like an elaborate process, but it works well even in places with no or difficult online connectivity. 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. A field correction requires another DigSig while verification might only report back to the inspector's back office. Pilfered plates will also be detected when used illegally or inserted back into the process. All of the role players can be held accountable because they can be identified through the DigSig they generated.

B.8 Selective struct

Often a Data Carrier is used for more than one group of data depending, for example, on the type of object it is attached to. The same DigSig may contain more than one data structure effectively describing more than one interface.

This is achieved by using the **nullable** feature. For example:

DigSig{PrimaryFields{}, FieldGroupX{}$^{\text{Nullable}}$, FieldGroupY{}$^{\text{Nullable}}$, FieldGroupZ{}$^{\text{Nullable}}$ }

In this case, the DigSig shall contain at least the PrimaryFields. It may contain the field groups X, Y and Z in any combination. The bit encoding of an excluded field group is the single bit 0$_2$.

EXAMPLE 1 CC…C$_2$: cid bits; SS…S$_2$: signature bits; TT…T$_2$: timestamp bits; bits; PP…P$_2$: PrimaryFields bits;

XX.Xx$_2$: FieldGroupX bits; YY..Y$_2$: FieldGroupY bits; ZZ..Z$_2$: FieldGroupZ bits

EXAMPLE 2 CC…CSS…STT…TPP…P000$_2$ – all field groups are excluded.

EXAMPLE 3 CC…CSS…STT…TPP…P0$_2$1$_2$YY…Y0$_2$ – field Y is included; field groups X and Z are excluded.

This also illustrates the compaction ability of the DigSigs method.

B.9 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$^{\text{QR}}${ContainerInfo{}, Source{}, Destination{}, SealTID{TagID}}

ContainerSeal^{63}{UII{SealInfo{}}, UserMem{~~Signature~~, Timestamp}, SealTID{TagID}}

B.10 Null encryption — Trusted data definition

Data verification is often not required. However, a requirement exists to have a well-defined interoperable data structure.

It is possible and feasible to specify no encryption to be used. The key length is zero and the length of the signature field will also be zero.

This means that the data encoding is exempt from a signature, which is typically quite large.

It is important to note that the DigSig Certificate remains a verifiable document and therefore the data structure and read methods are verifiable constructs. This is similar to an application obtained from the Internet which is digitally signed to ensure that it has not been tampered with.

B.11 Simple data error detection

The strong public key algorithms require large keys which result in large data structures. This potentially leads to slow RFID reads and large barcodes. In some applications, the use case does not demand high levels of security. The urge, in these cases, is to use shorter keys. The result is that the cryptographic strength is substantially reduced undermining the security which leads to a false sense of security.

A better solution can be the use of CRC-64 as the Hash algorithm with NULL encryption. This has the effect of a very good cyclic redundancy check (CRC) over the full data structure. The advantages are:

- No security is claimed and therefore no false sense of security. The signing method is published in the Certificate.
- No Digital Signature verification process is needed, resulting in a faster execution time.
- Data integrity errors are detected.

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 take action. 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 real world.

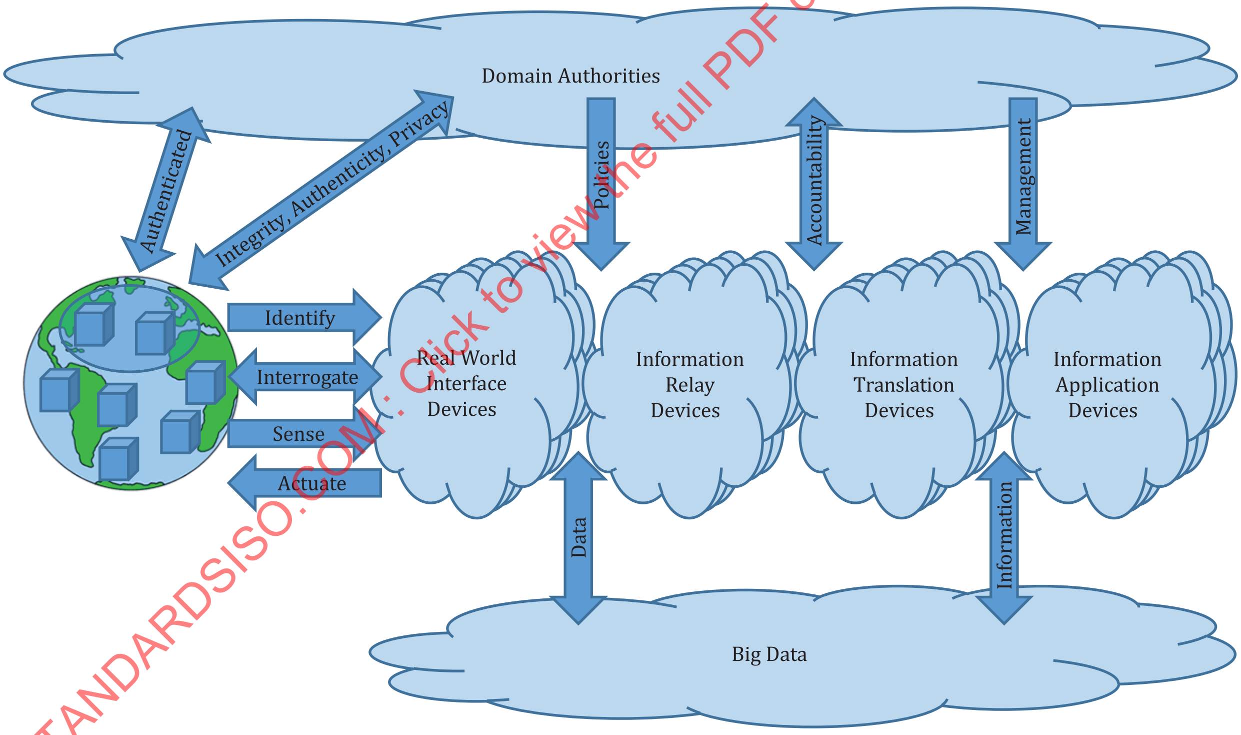

Domain Authorities
Authenticated
Integrity, Authenticity, Privacy
Policies
Accountability
Management
Identify
Interrogate
Sense
Actuate
Real World Interface Devices
Information Relay Devices
Information Translation Devices
Information Application Devices
Data
Information
Big Data

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

Automated identification data carriers fulfil an important role in representing real 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 also Figure 1).

- Real 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 is in the DDDdata format. See 8.2.2 and I.6.

- 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 PrivateInfo if it has been assigned authenticated access to the DigSig private container.

  The Real World Interface Device can also, by its own logic or under the directive of another device, write information to the Data Carrier.

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

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:

- Each and 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, each and 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 DigSig{AccessEventTimestamp, [AccessEventInfo], DigSig{ObjectID, ObjectInfo, PrivateInfo}}

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

  Any or all of the AccessEventInfo can be in a private container.

  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. It is recommended that such joining and aggregation also be signed.

— Following this chain, the Application device is able to 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 is 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

Figure D.1 illustrates a typical DigSig EncoderGenerator device architecture. The DigSig encoding and generation services typically consist of an Issuer device, a Trust Service 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 never be exposed from the EncoderGenerator device.

This Annex should be read with Annex E.

— The Issuer device is typically responsible for:

1) The assembly of the DigSig data to be signed,

2) The request for a DigSig,

3) The programming of the Data Carriers.

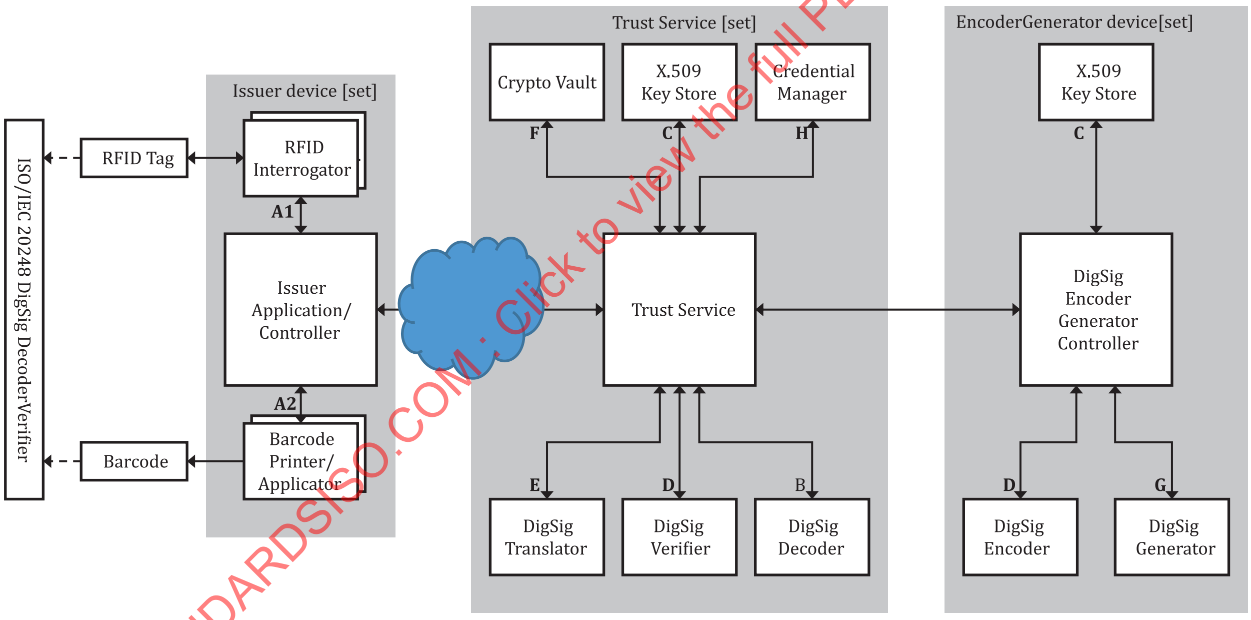


Figure D.1 — Typical DigSig EncoderGenerator use architecture

— The Trust Service is typically responsible for

1) The validation of the DigSig request and the credentials of the requestor (and issuing device),

2) The process of the DigSig generation,

3) The checking of the DigSig before release to the Issuer device,

4) Creating an audit trail of DigSig requests, DigSigs generated and error conditions,

5) It acts as a security shield for the EncoderGenerator device,

6) It may also provide the online verification services, and

   7) It may also provide the online DigSig revocation services.

- The EncoderGenerator device is typically responsible for
   1) The protection of the Private Key,
   2) The generation of the X.509 key pair,
   3) Journaling of all DigSigs generated,
   4) The generation of the Timestamp, SigData and Signature, and
   5) The encoding of the DigSig. This function may also be placed within the Trust Service.

This distributed architecture may be simplified if all the functions are local to a specific point, for example 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.

The DigSig encoding and generation process flow follows the description above.

Annex E
(informative)

Typical DigSig DecoderVerifier device architecture

This Annex should be read with Annex D.

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.

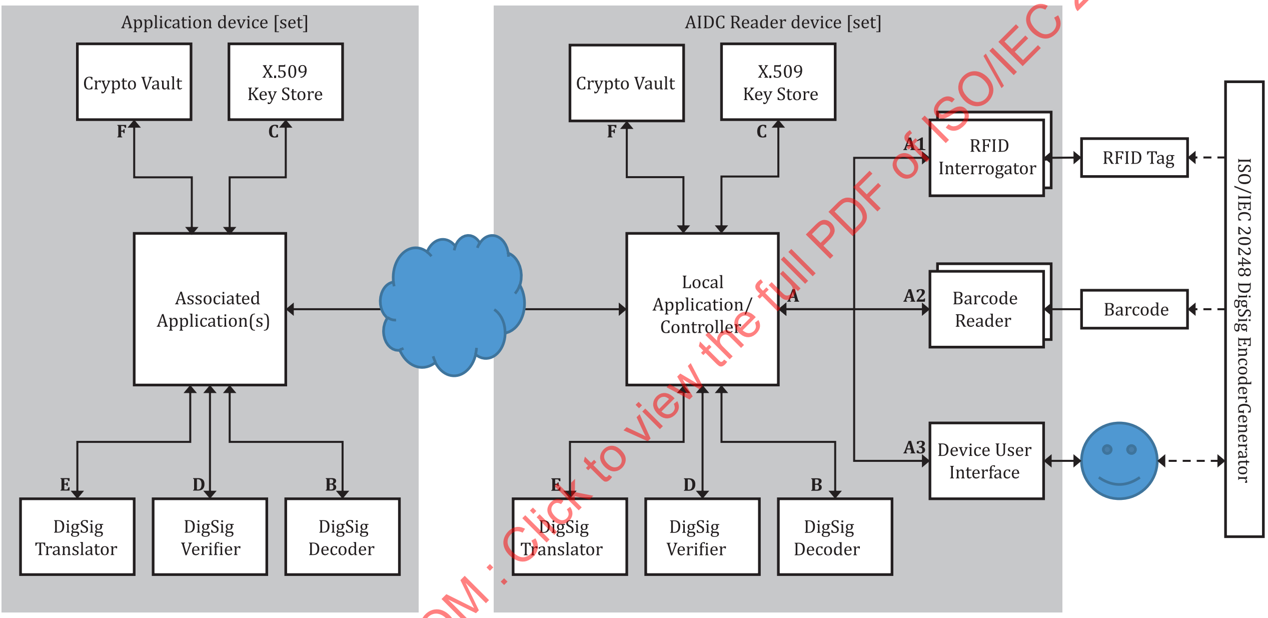


Figure E.1 — Typical DigSig DecoderVerifier use architecture

The architecture has the following modules.

a) 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.

b) 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.

c) RFID Interrogators: RFID Interrogators typically interrogate the Data Carrier providing direct and selective access to memory and over the air security.

d) Barcode Readers: Barcode readers typically read a barcode as a whole. They provide the data length and encoding information allowing the Application to interpret the data easily.

e) 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.

f) X.509 Key Store: The X.509 Key Store is typically a standards based component of the operating system. It stores and validates X.509 Certificates. The Application obtains a Certificate from a trusted source and stores it, after validation, in the X.509 Key Store. 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. This should be a function of the operating system. All modern mobile phones and personal computers have a X.509 Key Store.

g) 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, for example, used in POS terminals and ATMs.

h) 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 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.

i) DigSig Verification Module: The DigSig Verifier module verifies DDDdata using the referenced DigSig Certificate. It generates SigData from the DDDdata.

j) DigSig Translation Module: The DigSig DisplayInfo generates DDDdataTagged and DDDdataDisplay from DDDdata and the referenced DDD.

The process flow is as follows.

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

— The Local Application/Controller extracts the CID from the CIDSnip.

— 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.

— The Local Application/Controller extracts the DDD from the DigSig Certificate and passes it along with the CIDSnip to the DigSig Decoder.

— 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.

— The Local Application/Controller reads all the required DataSnips (**privatecontainer** DatatSnips are not decrypted at this stage). It then passes the array of Snips to the decoder for decoding.

— When the DDDdata are correctly decoded, the Local Application/Controller provides the DigSig Certificate and the DDDdata to the DigSig Verifier for verification.

— If the DigSig verification is accepted, the Local Application/Controller may request the DigSig Translator to generate DDDdataDisplay and DDDdataTagged for its own purposes.

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**.

— 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.

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 and 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.

Table F.1 *(continued)*

Code	Applicable	Error	Notes
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

A Digital Signature scheme typically consists of three algorithms.

a) 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 though remains secret. The public key is published in a Digital Certificate.

b) A signing algorithm (the hash and cryptographic functions) that, given a message and a private key, produces a signature.

c) 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:

a) It should be computationally infeasible to generate a valid Digital Signature if a party does not possess the private key.

b) The Digital Certificate 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 is an example of a DigSig Certificate to indicate where the DDD component is placed.

NOTE This example is only for illustration purposes. The content has been reduced to fit the page.

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 29 (0x1d)
  Signature Algorithm: sha256WithRSAEncryption
    Issuer: OU=Domain Control Validated, OU=EssentialSSL, CN=test.sigvr.it
    Validity
      Not Before: May 31 11:19:33 2017 UTC
      Not After: May 22 11:15:22 2018 UTC
    Subject: C=ZA, O=TrueVolve Internal Test Server,
CN=test.sigvr.it://41414/cid/2586582774
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
          04:60:49:81:a7:03:7d:3f:d7:01:d0:6d:21:32:0a:3a:0b:7f:91:e5:65:67:26:55:58:59:
          e7:38:ed:b6:0e:40:8b:0e:c2:36:d9:1d:fe:e6:63:2c:36:02:37:74:94:76:83:be:f0:9d:
          0b:53:d8:07:21:f9:10:02:ca:20:78:0a:eb
        ASN1 OID: prime256v1
    X509v3 extensions:
        X509v3 Key Usage: critical
            Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Server Authentication
        X509v3 Basic Constraints: critical
            CA:FALSE
        X509v3 Authority Key Identifier:
            keyid:1C:36:E5:4E:E4:AA:95:0C:50:4A:EB:6D:A5:07:44:97:9B:A2:C1:66
        X509v3 CRL Distribution Points:
            Full Name:
              URI:https://test.sigvr.it:41414/crl
        1.0.20248:
{"digsiginfo":{"specificationversion":"ISO/IEC 20248:2017","dauri":https://test.sigvr.it”
,"daid":"QC DGSG","cid":110,"verificationuri":"https://test.sigvr.it","revocationuri":"htt
ps://test.sigvr.it:41414/revoked"},"datafields":[{"fieldid":"specificationversion"},{"fiel
did":"dauri"},{"fieldid":"daid"},{"fieldid":"cid"},{"fieldid":"signature"},{"fieldid":"tim
estamp"},{"fieldid":"thefield","type":"string"},{"fieldid":"theNumberfield","type":"number
"}]}
  Signature Algorithm: sha256WithRSAEncryption
    c1:c6:46:54:1c:9f:0a:1d:30:97:64:99:bb:25:97:42:9f:0a:39:7f:ca:03:a4:70:86:98:6a:a5:
    ae:09:1b:be:c9:fc:d4:47:4d:e6:6e:d0:44:d6:b8:35:f3:36:2e:1f:91:77:38:a5:fc:60:54:af:
    e1:dd:d3:b5:5b:f7:ee:6c:75:90:8b:55:8e:11:cb:3a:be:7d:b6:a5:d6:66:26:14:de:f0:c3:8f:
    01:a9:1e:5b:1e:89:2a:86:47:2e:5a:cb:f8:b4:05:c3:af:f7:96:a9:dc:98:2b:3a:2c:2d:c0:66:
    36:0c:ec:1e:30:6e:f4:6e:1a:41:2c:ff:17:f9:cc:0b:21:ef:16:b8:57:a6:87:d9:38:3b:32:28:
    5a:13:7c:c1:ec:6a:1d:30:5a:cb:6d:f5:a5:21:2e:a6:47:72:f3:a9:5e:c4:ef:5d:79:9c:a6:91:
```

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.

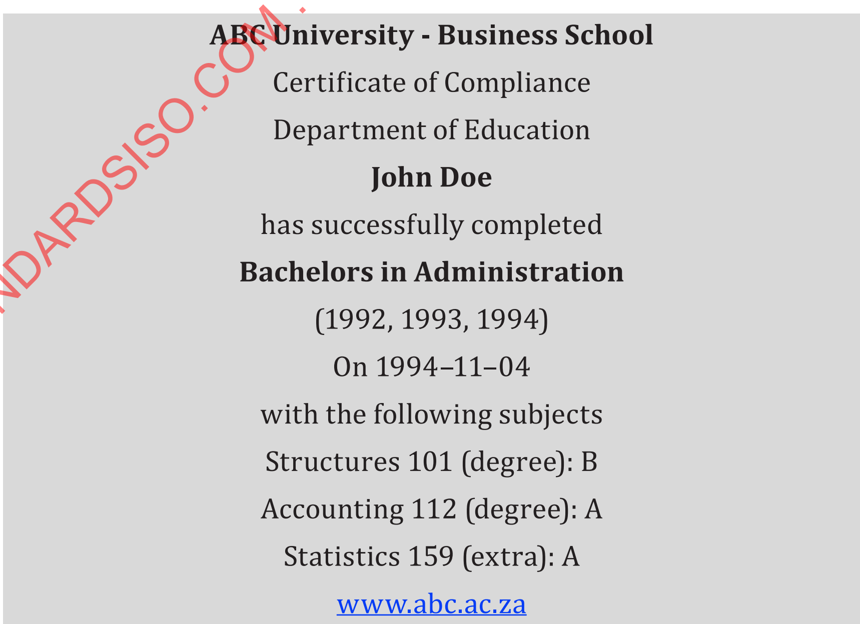
Two Data Carriers are used:

a) The first Data Carrier contains the information of the certificate excluding the certificate holder's identity number. It also contains a URL 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 a NFC tag. It uses the URI Envelope to support online verification typically with a standard mobile phone.

b) 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

(1992, 1993, 1994)

On 1994–11–04

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 20248:2017",
   "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":"https://www.dept-edu.com/uni-cert"
  },
 "datafields":
  [{"fieldid":"specificationversion"},
   {"fieldid":"dauri"},
   {"fieldid":"daid"},
   {"fieldid":"cid"},
   {"fieldid":"signature"},
   {"fieldid":"timestamp"},
   {"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":"unsignedint",
    "binaryformat":11,
    "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":"date",
    "fieldname":{"en":"Certificate Date","af":"Sertifikaat Datum"},
    "displayformat":"%Y-%m-%d"
   },
   {"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: “ekjcsaIOVbnNHSe7FM.........WqtMQNAtuwo3bgY=” in Base64url encoding

DataSnip: “612209498902” as a linear barcode using ISO/IEC 15417

The URI Envelope:

https://www.dept-edu.com?ekjcsaIOVbnNHSe7FM.........WqtMQNAtuwo3bgY=”

NOTE The Base64url_encoding(CIDSnip) is illustrative.

I.5 Example DDDdata input

```
["ISO/IEC 20248:2017","https://www.dept-edu.com","QC DGSG",110,null,null,
 "John Doe","612209498902",[1992,1993,1994],
 "Bachelors in Administration","Business School","www.abc.ac.za",
 783907200,
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]
 ]
]
```

I.6 Example DDDdata

```
 ["ISO/IEC 20248:2017","https://www.dept-edu.com","QC DGSG",110,
 "IOVbnNHSe7FM.........WqtMQNAtuwo3bgY=",783936000,
 "John Doe","612209498902",[1992,1993,1994],
 "Bachelors in Administration","Business School","www.abc.ac.za",
 783907200,
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]
 ]
]
```

NOTE The signature value is illustrative.

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 20248:2017","https://www.dept-edu.com","QC DGSG",110,783936000,"John Doe",
"612209498902",[1992,1993,1994],"Bachelors in Administration",783907200,[["Structures 101"
,"degree","B"],["Accounting 112","degree","A"],["Statistics 159","extra","A"]]]
```

SigData shall NOT contain white spaces or any other formatting. This below example is formatted for readability and comparison purposes.

```
["ISO/IEC 20248:2017","https://www.dept-edu.com","QC DGSG",110,
 783936000,
 "John Doe","612209498902",[1992,1993,1994],
 "Bachelors in Administration",
 783907200,
 [["Structures 101","degree","B"],
  ["Accounting 112","degree","A"],
  ["Statistics 159","extra","A"]
 ]
]
```

I.8 Example DDDdataTagged

```
{"specificationversion":"ISO/IEC 20248:2017",
 "dauri":"https://www.dept-edu.com",
 "daid":"QC DGSG",
 "cid":110,
 "signature":"IOVbnNHSe7FM.........WqtMQNAtuwo3bgY=",
 "timestamp":783936000,
 "studentname":"John Doe",
 "idnumber":"612209498902",
 "years":[1992,1993,1994],
 "coursename":"Bachelors in Administration",
 "department":"Business School",
 "departmentURL":"www.abc.ac.za",
 "date":783907200,
 "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 20248:2017",
   "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 20248:2017"
   },
   {"fieldid":"dauri",
    "displayvalue":"https://www.dept-edu.com"
   },
   {"fieldid":"daid",
    "displayvalue":"QC DGSG"
   },
   {"fieldid":"cid",
    "displayvalue":"110"
   },
   {"fieldid":"signature",
    "displayvalue":"IOVbnNHSe7FM.........WqtMQNAtuwo3bgY="
   },
   {"fieldid":"timestamp",
    "displayvalue":"1994-11-04T08:00UTC"
   },
   {"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":["1992","1993","1994"]
   },
   {"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":"1994-11-04"
   },
   {"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

The Domain Authority shall use the following CIDs for all its DigSig revocations.

— CID ⇐ 0: DigSig revocation list DDD; see J.4.

— CID ⇐ 1: DigSig revocation check DDD; see J.5.

J.2 Revocation response rules

The DigSig of which its revocation status is to be checked shall be referenced by {<DAID>, <CID>}.

The hash algorithm used in the list and for the revocation response DigSigs shall be the hash algorithm specified in the DigSig Certificate {<DA>, <CID>} which describes the revoked DigSigs.

NOTE Revoking null encryption DigSigs (see B.10) does not make sense; however, if done, the encryption method at least contains a hash function as specified in B.11.

The **signature** and **timestamp** field values (see 6.3 and 8.5) of the revocation response DigSig shall be generated during the signing of the revocation response DigSig.

J.3 DigSig revocation lists

A DigSig revocation list shall contain the serialised timestamped list of the revoked DigSigs with the following record format:

> {serial number, revocation timestamp, Hash(SigData)}

The serialisation shall start with the number one.

J.4 DigSig revocation list download

A DigSig revocation list shall be requested using the following URI:

> "<revocationuri>?daid=<DAID>&cid=<CID>[&locallistend=<The sequence number of the last record in the list at the requester for this {<DA>, <CID>.}>[&downloadlimit=<Maximum amount of record in download.>]"

The server has a veto on the download limit. If **downloadlimit** is not specified, the server decides.

If **locallistend** is not provided then the list will be downloaded from its start.

The revocation list shall be returned in a special DigSig with the following **datafields**:

```
"datafields":
 [{"fieldid":"specificationversion"},
  {"fieldid":"dauri"},
  {"fieldid":"daid"},
  {"fieldid":"cid"},
  {"fieldid":"signature"},
  {"fieldid":"timestamp"},
    {"fieldid":"lastlistpart","type":"boolean"},
    {"fieldid":"revocationlistpart",
     "type":"struct",
     "nullable":true,
     "cardinality":"variable",
     "fields":
       [{"fieldid":"sequence","type":"unsignedint"},
        {"fieldid":"revocationtimestamp","type":"date"},
        {"fieldid":"digsighash","type","bstring"}
       ]
   ]
```

EXAMPLE A DDDdata revocation list result for the DDD {“https://www.dept-edu.com”, 110} is as follows:

```
 ["ISO/IEC 20248:2017","https://www.dept-edu.com","QC DGSG",110,
  "IOVbnNHSe7FM.........WqtMQNAtuwo3bgY=",783936000,true,
  [[1,784783233,"lkjlkajf7fsakhw982h3hd298"],
   [2,784783456,"kajslfofeq870pjrpf80jcjcl"],
   [3,784784323,"123jrpo932fjlwe9832jfa99p"],
   [4,784784539,"fkaj2398r2qpf09j432jf90fc"]
  ]
 ]
```

The values are illustrative. The DDDdata are formatted for readability purposes.

J.5 Online DigSig revocation check

An online DigSig revocation check shall be performed using the following URI:

“<revocationuri>?daid=<DAID>&cid=<CID>&sigdatahash=<Hash(SigData)>”

The result of “valid”, “revoked” or “error” shall be returned in a special DigSig with the following **datafields**:

```
 "datafields":
  [{"fieldid":"specificationversion"},
   {"fieldid":"dauri"},
   {"fieldid":"daid"},
   {"fieldid":"cid"},
   {"fieldid":"signature"},
   {"fieldid":"timestamp"},
   {"fieldid":"revocationresult",
    "type":"enum",
    "enumvalues":["valid","revoked","error"]}
  ]
```

EXAMPLE A DDDdata revocation result for a DigSig of the DDD {“https://www.dept-edu.com”, 110} is as follows:

```
 ["ISO/IEC 20248:2017","https://www.dept-edu.com","QC DGSG",110,
  "IOVbnNHSe7FM.........WqtMQNAtuwo3bgY=",783936000,"valid"]
```

The values are illustrative. The DDDdata are formatted for readability purposes.

Annex K
(normative)

2D bar code symbologies — Encoding and decoding the DigSig

K.1 Symbologies capable of supporting DigSigs

The DigSig may be encoded in any of the following 2D bar code symbologies:

- Aztec Code, as defined in ISO/IEC 24778;
- Data Matrix, as defined in ISO/IEC 16022;
- MicroPDF417, as defined in ISO/IEC 24728;
- PDF417, as defined in ISO/IEC 15438;
- QR Code, as defined in ISO/IEC 18004.

K.2 Requirements and constraints

K.2.1 Symbology encoder requirements and constraints

The symbology encoder shall support the ISO/IEC 15434 message structure, in particular Format "06" for ASC MH 10 Data Identifier for the RAW envelope. The ISO/IEC 15434 rules that apply to this document are fully defined in K.3.1 and K.3.2.

The URI envelope encoding scheme (see K.4) is the exception to the ISO/IEC 15434 message structure.

Where a symbology supports a choice of error correction, the encoding system shall be capable of supporting the rules defined in K.5.

This document imposes no constraints on the following features, leaving this to the domain authority to define based on the requirements of the application:

- the choice of the symbology from those defined in K.1;
- the size and format of the symbol from those specified for the symbology of choice;
- the use of structured append might cause problems, so it should be adopted with care;
- the x-dimension, as long as it meets basic application requirements;
- the use of a higher error correction level defined in K.5.

NOTE In normal circumstances, structured append can be used to perfectly reconstruct data encoded over more than one 2D symbol. The use of an ISO/IEC 15434 message and the fact that some DataSnips might be associated with a particular read instruction specified by the pragma calls for structured append to be used with some care.

K.2.2 Symbology decoder and reader requirements and constraints

A standard symbology decoder can be used to support the reading of an ISO/IEC 15434 message, but the post-decode data processing shall be capable of decoding such a message and parsing it to extract the data value associated with the ASC MH 10.8 Data Identifier 6R.

The URI envelope encoding scheme (see K.4) is the exception to the ISO/IEC 15434 message rule. A compliant URI symbology decoder does not need to support this message structure and its post decode processing, but may do so if other types of DigSig are defined for the application domain.

If the selected error correction fails and the message cannot be properly reconstructed, then an appropriate error message shall be transmitted to the application.

K.3 Common RAW envelope encoding and decoding rules

K.3.1 RAW envelope encoding

The RAW envelope is defined in 8.3.3.2. This envelope structure is used to support DigSigs in applications that already make use of 2D symbols in existing applications. When encoded in the 2D symbol, the RAW envelope is the data value of Data Identifier 6R. This is constructed, together with other Data Identifiers and associated data, in an ISO/IEC 15434 message.

Additional data elements may be specified as part of the DigSig and be encoded in other Data Identifiers.

The encoding process is undertaken in these steps.

a) The DigSig EncoderGenerator transmits the RAW envelope and Data Identifier in the form of the 6R with its value to the ISO/IEC 15434 encoding process.

b) If relevant, the DigSig EncoderGenerator transmits related Data Identifier(s) and the associated data to the ISO/IEC 15434 encoding process.

c) If relevant, the application transmits other Data Identifier(s) and the associated data to the ISO/IEC 15434 encoding process.

d) Application-defined rules are used to define the sequence of Data Identifiers

e) The ISO/IEC 15434 Format “06” message is constructed.

f) The ISO/IEC 15434 message is encoded to the specific symbology rules.

g) The symbology encoding process shall employ error correction as appropriate for the selected symbology (see K.5).

h) (Implied) all the other 2D symbol parameters (e.g. choice of symbol size and error correction) are processed before the symbol is printed.

K.3.2 RAW envelope decoding

The specific symbology decoder removes any error correction and other overhead features to its own decoding rules.

The resultant byte string is transferred as a complete ISO/IEC 15434 Format “06” message to a parsing process. If this contains Data Identifier 6R, the DigSig DDD shall be used to identify other Data Identifiers and the associated data that are part of the DigSig.

K.4 URI envelope encoding and decoding rules

K.4.1 URI envelope encoding

The URI envelope is defined in 8.3.3.3. The objective is to make use of the large scale deployment of hand-held devices with generic 2D symbology decoders, e.g. in smartphones. When encoded in the 2D symbol, the URI envelope comprises, in this sequence:

a) The verificationuri to enable this to be processed by any commonly available browser system.

b) The character “?”.

c) The CIDSnip, where the DigSig EncoderGenerator will have selectively used the Base64url encoding scheme to render all characters as eye readable.

d) Optionally the character “~” and the cidsniptextvalue.

The encoding follows the rules of the selected 2D symbology, using the appropriate mode and shift characters to achieve an efficient encoding. This produces a fairly basic DigSig which can be deployed on a large number of applications using low-cost reading devices.

The encoding process shall employ error correction as appropriate for the selected symbology (see K.5).

K.5 Symbology specific rules

K.5.1 Common rules for all symbologies

Constructing the ISO/IEC 15434 message, and symbology specific compaction, form two independent logical layers of the process for the symbologies described in this annex.

— The ISO/IEC 15434 message is considered a separate layer in the systems stack from the rules defined in the specific symbology standard. Similarly, the ISO/IEC 15434 message shall be reconstructed after the symbology decoding procedure. These symbology rules are defined in the appropriate standard mentioned below.

— Once constructed, the ISO/IEC 15434 message string is not necessary “compacted” as a single byte. A given string of data bytes may be represented by different codeword sequences, depending on how the encoder switches between compaction modes and sub-modes. There is no single specified way to encode data in a 2D symbol. For example, the byte string 30:31:32:34:39:37:32:31:34:36:31 might well be considered suitable for encoding using the numeric compaction mode of the specific symbology. This mode switching is invoked by mechanisms similar to shift and lock on a keyboard and generally results in fewer data codewords being required. During the decode process, the mechanisms that declare the mode switch are identified and the original byte string is reconstructed.

The implication of these two points is that the symbology data compaction schemes can reduce the encoding. In turn, it means that the DigSig cannot be decoded directly from the symbology decoder, but from the ISO/IEC 15434 message structure.

Different symbologies have procedures to concatenate the data from different symbols into one message. The requirement is to maintain the structure of the entire ISO/IEC 15434 message; therefore, in this first edition of this document, the various forms of structured append should not be used.

K.5.2 ISO/IEC 15438 PDF417

K.5.2.1 Common rules for PDF417

Compact PDF417 shall not be used for DigSigs.

Macro PDF417 should not be used for the reasons defined in K.5.1.

PDF417 support 9 different levels of error correction. Minimum error correction levels are defined in ISO/IEC 15438 for different numbers of data codewords. A higher level of error correction may be specified by the application.

K.5.2.2 PDF417 and RAW envelopes

All encoding of the ISO/IEC 15434 message is based on the PDF417 mode switching rules.