**S-100 – Part 15**

**Data Protection Scheme**

Page intentionally left blank

**Contents**

15-1 Scope 1

15-2 Normative References 1

15-3 General Description 1

15-4 Participants in the Protection Scheme 2

15-4.1 Scheme Administrator 2

15-4.2 Data Servers 3

15-4.3 Data Clients 3

15-4.4 Original Equipment Manufacturers 3

15-4.5 Domain Coordinator 3

15-4.6 Participant Relationships 3

15-5 Data compression and packaging 4

15-5.1 Overview 4

15-5.2 Compression Algorithm 5

15-5.3 Encoding 5

15-6 Data encryption 5

15-6.1 What Data is encrypted? 5

15-6.2 How is it encrypted? 5

15-6.2.1 Encryption algorithm 5

15-6.2.2 Encryption padding 6

15-6.2.3 AES encryption CBC mode 6

15-6.2.4 AES CBC mode – initialization vector 7

15-6.2.5 AES examples 8

15-7 Data encryption and licensing 8

15-7.1 Introduction 8

15-7.2 Conversion of bit strings to integers 9

15-7.2.1 Converting bit strings to an integer 9

15-7.2.2 Converting an integer number to a bit string 10

15-7.2.3 Converting an unsigned integer number to a hexadecimal text representation 10

15-7.2.4 Converting a hexadecimal text representation to an unsigned integer number 11

15-7.3 The User Permit 11

15-7.3.1 Definition of user permit 12

15-7.3.2 M\_KEY Format 12

15-7.4 The data permit 13

15-7.4.1 The permit file (PERMIT.XML) 13

15-7.4.2 The Permit File - Header content 14

15-7.4.3 Product sections and permit records fields 15

15-7.4.4 Definition of the permit record 15

15-7.4.5 Permit file signatures 16

15-7.4.6 An example PERMIT.XML file 16

15-8 Data authentication 17

15-8.1 Introduction to data authentication and integrity checking 17

15-8.2 Data Protection Scheme setup, Data Server signup and authentication sequence 18

15-8.3 Verification of digital signatures 19

15-8.4 Data Formats and standards for digital signatures, keys and certificates 19

15-8.5 Creation of key material and certificate signing requests (signed Public Keys) 21

15-8.5.1 SA setup 21

15-8.5.2 Data Server setup 21

15-8.6 Digital certificate example 22

15-8.7 Creation of digital signatures by a Data Server 22

15-8.8 Additional digital signatures 23

15-8.9 Verifying Data Integrity and Digital Identity with an S-100 digital signature 24

15-8.10 MRN specifications 25

15-8.11 Exchange catalogue metadata and standalone schema element specification 27

15-8.11.1 S100\_SE\_CertificateContainerType 28

15-8.11.2 StandaloneDigitalSignature 28

15-8.11.3 S100\_SE\_DigitalSignature 28

15-8.11.4 S100\_SE\_SignatureOnData 28

15-8.11.5 S100\_SE\_SignatureOnSignature 29

15-8.11.6 DataStatus 29

15-8.11.7 S100\_SE\_DigitalSignatureReference 29

15-9 Glossary of S-100 Data Protection Scheme and computing terms 31

# Scope

S-100 part 15, later referred to as ‘the Data Protection Scheme’ or ‘Protection Scheme’, describes the recommended standard for the protection of hydrographic or spatial information based on the IHO S-100 Universal Hydrographic Data Model. It defines security constructs and operating procedures that must be followed to ensure that the Protection Scheme is operated correctly and to provide specifications that allow participants to build compliant systems and distribute data in a secure and commercially viable manner.

# Normative References

The following referenced documents are required for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including amendments) applies.

FIPS Publication 81, *DES Modes of Operation*, National Institute of Standards and Technology <[www.itl.nist.gov/fipspubs/fip81.htm](http://www.itl.nist.gov/fipspubs/fip81.htm)>

FIPS Publication 180-4, *Secure Hash Standard (SHS)*

 <<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>>

FIPS Publication 186, *Digital Signature Standard (DSS)* <[www.itl.nist.gov/div897/pubs/fip186.htm](http://www.itl.nist.gov/div897/pubs/fip186.htm)>

ISO/IEC 18033-3, *Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers (AES)*

*Open SSL Cryptography and SSL/TLS Toolkit* <<https://www.openssl.org/>>

PKCS#10 v1.7, *Certification Request Syntax Specification* <<https://tools.ietf.org/html/rfc2986>>

RFC 1423, *Privacy Enhancements for Internet Electronic Mail: Part III: Algorithms, Modes and Identifiers* <ftp://ftp.isi.edu/in-notes/rfc1423.txt>

RFC 2451, *The ESP CBC-Mode Cipher Algorithms* <<https://tools.ietf.org/html/rfc2451>>

RFC 2459 version 3, *Internet X.509 Public-key infrastructure and attribute certificate frameworks* <<https://tools.ietf.org/html/rfc2459>>

RFC 5651, *Cryptographic Message Syntax (CMS)*, ITU International Telecommunication Union <<https://tools.ietf.org/html/rfc5652#section-6.3>>

RFC 4647, Base 64 Encoding. <<https://datatracker.ietf.org/doc/html/rfc4648#section-4>>

OSI networking and system aspects – Abstract Syntax Notation One (ASN.1), ITU International Telecommunication Union <<https://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf>>

X.509 Version 3, *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework*, International Telecommunication Union

# General Description

This Part specifies a method of securing digital nautical, hydrographic and spatial related products and information. The purpose of data protection is threefold:

1. Piracy Protection: To prevent unauthorized use of data by encrypting the product information.
2. Selective Access: To restrict access to only the products that a customer has acquired a license for.
3. Authentication: To provide assurance that the products have come from approved sources.

Piracy protection and selective access are achieved by encrypting the products and providing data permits to decrypt them. Data permits have an expiration date to enable access to the products for a licensed period. Data Servers will encrypt digital products before supplying them to the Data Client. The encrypted products are then decrypted by the end-user system (for example ECDIS/ECS) prior to use. Authentication is provided by means of digital signatures applied to the product files.

The security scheme does not address how the product information is protected once it is within an end-user application. This is the responsibility of the Original Equipment Manufacturers (OEMs).

The Scheme enables the mass distribution of protected datasets on hard media which can then be accessed and used by all customers with a valid license containing a set of data permits. Selective access to individual products is supported by providing users with a licensed set of data permits containing the encrypted dataset keys. This license is created using a hardware identifier of the target system and is unique to each Data Client. Consequently licenses cannot be exchanged between individual Data Clients.

The Protection Scheme is designed for file based transfer of data between parties. Stream based transfer may use different methodologies. Data streaming is presented in S-100 Part 14. The S-100 Protection Scheme described in this Part is bound to the value “S100p15” in the protectionScheme element of the CATALOG.XML Exchange Set Catalogue.

The Scheme uses an optional compression algorithm to reduce the size of the dataset. Unencrypted product files contain many repeating patterns of information; for example coordinate information. Compression is therefore always applied before the product file is encrypted and uncompressed after the corresponding decryption on the data client system.

# Participants in the Protection Scheme

There are several types of users of the Scheme; these are as follows:

* The Scheme Administrator (SA), of which there is only one;
* The Data Server (DS), of which there can be many;
* The Data Client (DC), of which there are many;
* The Original Equipment Manufacturer (OEM) of which there are many;
* Domain Coordinators, of which there may be many.

A more detailed explanation of these terms is given below. Details of the roles for each of the scheme participants are managed by the IHO acting as the Scheme Administrator.

## Scheme Administrator

The Scheme Administrator (SA) is solely responsible for maintaining and coordinating the Protection Scheme. The SA role is operated by The International Hydrographic Organization (IHO) on behalf of the IHO Member States and other organizations participating in the Protection Scheme. These organizations can have a coordinating role for a maritime product domain; for example IMO and IALA. The IHO as the SA will establish procedures with product domain operators using the Protection Scheme to protect their products. These procedures will enable these domain coordinators to digitally sign the digital certificates used by their member organizations to participate in the Protection Scheme.

The SA is responsible for controlling membership of the Scheme and ensuring that all participants operate according to defined procedures. The SA maintains the top level digital root certificate used to operate the Protection Scheme and which forms the root identity in the authentication chain.

The SA is responsible for distributing the manufacturer ID (M\_ID) and manufacturer key (M\_KEY) directly to all registered Data Servers participating in the Protection Scheme.

The SA is also the custodian of all documentation relating to the implementation of this part of S-100. All operational procedures are defined and managed by the SA.

## Data Servers

Data Servers (DS) are responsible for the encryption and/or digital signing of the datasets in compliance with the procedures and processes defined in this Part. Data Servers may also issue Licenses (data permits) so that Data Clients, with valid user permits, can decrypt the product data.

Data Servers will use the M\_KEY and M\_ID information, as supplied by the SA, to issue encrypted product keys to each specific installation. Even though the keys used to encrypt each dataset are the same for individual data clients, they will be encrypted using the HW\_ID and therefore cannot be transferred between other system installations from the same manufacturer.

The Scheme does not impede agents or distributors from providing data services to their customers. Agreements and structures to achieve this are outside the scope of this document. This document contains only the technical specifications to produce protected datasets compliant with this standard.

Hydrographic Offices, data producers, Value Added Resellers and RENC Organizations are examples of Data Servers.

## Data Clients

Data Clients (DC) are the end users of datasets and will receive protected information from the Data Servers to access and use the datasets and services. The Data Client’s software application (OEM System) is responsible for authenticating the digital signatures applied to the product files and decrypting the dataset information in compliance with the procedures defined in the Scheme.

Navigators with ECDIS/ECS systems are examples of Data Clients.

## Original Equipment Manufacturers

Original Equipment Manufacturers (OEMs) subscribing to the S-100 Data Protection Scheme must build a software application according to the specifications set out in this document and self-verify and validate it according to the terms mandated by the SA. This Part will establish test data for the verification and validation of OEM applications for various S-100 based Product Specifications when products become available. The SA will provide successful OEM applicants with their own unique manufacturer key and identification (M\_KEY and M\_ID).

The manufacturer must provide a secure mechanism within their software systems for uniquely identifying each end user installation. The Scheme requires each installation to have a hardware identifier (HW\_ID) unless a Data Server has consented to duplication.

The software application will be able to decrypt the product keys in the data permits using the HW\_ID stored in either the hard lock or soft lock devices attached to or programmed within the application to subsequently decrypt and uncompress the dataset files. Product integrity can be verified by authenticating the digital signature provided with the dataset files.

## Domain Coordinator

Domain Coordinators are nominated, trusted bodies of the SA, able to produce certificates and provide intermediate authentication of Data Servers within their domain. Domain Coordinators are appointed by the SA and have delegated authority to sign Data Server certificates within their own domain. When Data Clients authenticate the identity of digital signatures created by Data Servers the certificates form a “chain” to the SA’s root level identity. If a Data Server is certified by a Domain Coordinator, then the Data Client should also verify their identity against the SA root providing an authentication chain from the dataset to the SA.

## Participant Relationships

The Scheme Administrator (SA), of which there can only be one, authenticates the identity of the other participants within the scheme. All Data Servers, Domain Coordinators and System Manufacturers (OEMs) must apply to the SA to become participants in the Scheme and, on acceptance, are supplied with proprietary information unique to them. Data Clients are customers of Data Servers and OEMs, where Data Servers supply data services and OEMs the equipment to decrypt and display these services.

The SA will sign the public key of Data Servers to create their digital certificate to be used in the operation of the Protection Scheme. It is also possible for Domain Coordinators to sign the public key of their member organizations to create their digital certificates. The Domain Coordinators will inform the SA of each Data Server’s identity and contact details and sign their certificates. The SA and Domain Coordinators will distribute M\_ID and M\_KEY information directly to all Data Servers participating in the Protection Scheme when they join the scheme and as more Data Clients are added.



Figure 15-1 – Relationship between Protection Scheme participants

Since the Protection Scheme does not rely on Data Clients always having an internet connection to authenticate certificates or for certificate path validation, sufficient information shall be included in the Exchange Set Metadata to perform these functions. In all cases the SA certificate is installed on end user systems separately and not distributed in the exchange set metadata to provide independent verification of the SA certificate.

# Data compression and packaging

## Overview

The content of products based on the S-100 Data Model will, because of their structure, contain repeating patterns of information. Examples of this are small variations in the coordinate information within the file.

If compression is applied, the files are always compressed before they are encrypted as the effectiveness of any compression algorithm relies on the existence of structured data contents. The individual S-100 based Product Specifications will specify in metadata whether compression is being used.

**All exchange set files must be digitally signed before any compression is applied.**

## Compression Algorithm

The Protection Scheme uses the ZIP algorithm to compress and uncompress files. The compression method is DEFLATE. Each file is compressed into a single file archive with the same name as the source file. If it is required to compress multiple files (for example, a Portrayal Catalogue) then they shall be located in a single root folder and the name of the compressed file set to the name of the root folder.

The encryption and digital signature features of ZIP are not used.

## Encoding

The individual S-100 based Product Specifications will provide more details if compression is being used, and which files will be compressed.

The use of compression will be encoded:

* S-100\_ExchangeCatalogue-compressionFlag with value **1**.

# Data encryption

## What Data is encrypted?

Any Product Specification that is based on the S-100 Data Model must define whether encryption will be used and which files will be encrypted.

When encrypted, the encryption algorithm must be the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode of operation. It is always assumed that the complete file will be encrypted.

In addition the OEM System HW\_ID (hardware ID) will be encrypted and provided to the Data Client in the form of a user permit. The keys used to encrypt the files are themselves encrypted by the Data Server and supplied to Data Clients as data permits. Information about the encryption algorithm is available in clause 15-6.2.1.

## How is it encrypted?

Each single product is encrypted using a unique key. The same key is used to encrypt all files associated with the product and all updates issued for the product edition. The Scheme, however, allows for the keys to be changed at the discretion of the Data Server. The keys are delivered to Data Clients in the form of data permits.

### Encryption algorithm

For encryption of permits and data files the Advanced Encryption Standard (AES) block cipher algorithm is used. This is a symmetric-key algorithm. This means that the same key is used for encryption and decryption. The algorithm defines how one block of plain text is converted to one block of cipher text and vice versa. The block size of the AES is always 16 Bytes (128 bit). The key length can be chosen from 128 bit, 192 bit or 256 bit. The corresponding variants are named AES-128, AES-192, or AES-256. In this Part of S-100 a 128 bit key length is always used.

The AES algorithm can only encrypt one block of plain text. For larger messages a block cipher mode of operation shall be used. This Protection Scheme chooses the Cipher Block Chaining (CBC) mode for encryption of more than one block of data. In this mode of operation it is required that the length of the plain text must be an exact multiple of the block size; padding is required.

### Encryption padding

The padding methods that will be used is described in PKCS#7. It adds N bytes to the message until its length is a multiple of 16 Bytes. The value of each byte is N. Note that if the original plain text has already a multiple of 16 as length a full block of 16 bytes each having the value of 16 must be added.

Table 15-1 – Plain text padding

|  |  |
| --- | --- |
| Plain text | Padded plain text |
| xx  | xx **0F 0F 0F 0F 0F 0F 0F****0F 0F 0F 0F 0F 0F 0F 0F** |
| xx xx | xx xx **0E 0E 0E 0E 0E 0E****0E 0E 0E 0E 0E 0E 0E 0E** |
| xx xx xx | xx xx xx **0D 0D 0D 0D 0D****0D 0D 0D 0D 0D 0D 0D 0D** |
| xx xx xx xx | xx xx xx xx **0C 0C 0C 0C****0C 0C 0C 0C 0C 0C 0C 0C** |
| xx xx xx xx xx  | xx xx xx xx xx **0B 0B 0B****0B 0B 0B 0B 0B 0B 0B 0B** |
| xx xx xx xx xx xx | xx xx xx xx xx xx **0A 0A****0A 0A 0A 0A 0A 0A 0A 0A** |
| xx xx xx xx xx xx xx  | xx xx xx xx xx xx xx **09****09 09 09 09 09 09 09 09** |
| xx xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xx**08 08 08 08 08 08 08 08** |
| xx xx xx xx xx xx xx xxxx | xx xx xx xx xx xx xx xxxx **07 07 07 07 07 07 07** |
| xx xx xx xx xx xx xx xxxx xx | xx xx xx xx xx xx xx xxxx xx **06 06 06 06 06 06** |
| xx xx xx xx xx xx xx xxxx xx xx | xx xx xx xx xx xx xx xxxx xx xx **05 05 05 05 05** |
| xx xx xx xx xx xx xx xxxx xx xx xx | xx xx xx xx xx xx xx xxxx xx xx xx **04 04 04 04** |
| xx xx xx xx xx xx xx xxxx xx xx xx xx | xx xx xx xx xx xx xx xxxx xx xx xx xx **03 03 03** |
| xx xx xx xx xx xx xx xxxx xx xx xx xx xx | xx xx xx xx xx xx xx xxxx xx xx xx xx xx **02 02** |
| xx xx xx xx xx xx xx xxxx xx xx xx xx xx xx | xx xx xx xx xx xx xx xxxx xx xx xx xx xx xx **01** |
| xx xx xx xx xx xx xx xxxx xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xxxx xx xx xx xx xx xx xx**10 10 10 10 10 10 10 10****10 10 10 10 10 10 10 10** |

**xx** = Arbitrary Bytes

### AES encryption CBC mode

In CBC mode each block of plain text is XORed with the previous cipher text block before being encrypted. An initialization vector IV is required for the first block. The mathematical formula is:

$C\_{i}=E\_{K}\left(P\_{i }⊕C\_{i-1}\right); i\geq 1$ (3a)

$C\_{0}=IV$ (3b)

*Ci* is the ith block of cipher text; *Pi* is the ith block of plain text. *EK* is the encryption method of AES encrypting exactly one block. *IV* is the initialization vector, and $⊕$ is the XOR operation.



Figure 15-2 – Cipher Block Chaining (CBC) mode encryption (Source: Wikipedia)

Decryption is defined as:

$P\_{i}=D\_{K}\left(C\_{i}\right)⊕C\_{i-1}; i\geq 1$ (4a)

$C\_{0}=IV$ (4b)

*DK*is the decryption method of AES decrypting exactly one block.



Figure 15-3 – Cipher Block Chaining (CBC) mode decryption (Source: Wikipedia)

### AES CBC mode – initialization vector

Normally the initialization vector must be transferred from the encryption to the decryption. However an incorrect IV at the decryption will only corrupt the first plain text block. This can be easily recognized from the formulas and the diagrams. Each plain text block depends only on two adjacent cipher text blocks.

This behaviour will be used in the following modification of the CBC mode.

On encryption of data files the plain text will be prepended by a single random block. Then encryption is done as normal using a random initialization vector. This vector does not have to be transferred to the decryption at the Data Client.

On decryption an arbitrary initialization vector can be used and after normal CBC decryption the first plain text block is discarded. The rest is the original plain text data file.

This procedure does not require the transport of the IV or the use of a predicted IV. The first option would complicate the process of data transfer and the second would make it vulnerable to attacks especially if the first blocks of plain text are commonly known (as ISO/IEC 8211 Data Descriptive Records).

### AES examples

The following examples are taken from the FIPS documentation.

Encrypting and decrypting of exactly one block:

Key128: K = {00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f}

Plain Text: P = {00, 11, 22, 33, 44, 55, 66, 77, 88, 99, aa, bb, cc, dd, ee, ff}

Cipher Text: C = {69, c4, e0, d8, 6a, 7b, 04, 30, d8, cd, b7, 80, 70, b4, c5, 5a}

Key192: K = {00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f,

 10, 11, 12, 13, 14, 15, 16, 17}

Plain Text: P = {00, 11, 22, 33, 44, 55, 66, 77, 88, 99, aa, bb, cc, dd, ee, ff}

Cipher Text: C = {dd, a9, 7c, a4, 86, 4c, df, e0, 6e, af, 70, a0, ec, 0d, 71, 91}

Key256: K = {00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f,

 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f}

Plain Text: P = {00, 11, 22, 33, 44, 55, 66, 77, 88, 99, aa, bb, cc, dd, ee, ff}

Cipher Text: C = {8e, a2, b7, ca, 51, 67, 45, bf, ea, fc, 49, 90, 4b, 49, 60, 89}

The following example documents the modified CBC mode:

Key128: K = {12, 34, 56, 78, 9a, bc, de, f0, 12, 34, 56, 78, 9a, bc, de, f0}

Plain Text: P = {fe, dc, ba, 98, 76, 54, 32, 10}

Plain Text after prepending a random block:

P’ = {48, d2, 4e, 7c, 00, 2f, 67, 4e, 93, 1d, ee, 27, 42, 17, a3, 4c}
 {fe, dc, ba, 98, 76, 54, 32, 10}

Plain Text (padded):

*P” =* {48, d2, 4e, 7c, 00, 2f, 67, 4e, 93, 1d, ee, 27, 42, 17, a3, 4c}
 {fe, dc, ba, 98, 76, 54, 32, 10, 08, 08, 08, 08, 08, 08, 08, 08}

Initialization vector (random):

IVE = {45, b5, 00, d7, 28, 39, 42, bb, 85, 61, 28, d5, 97, 15, ca, 25}

Cipher Text using CBC Mode:

C = {ba, 45, ee, 06, 02, a6, 29, 35, 7a, e3, 90, 2c, 22, 4d, d9, d5}
 {dd, 3b, 07, 3b, 84, 7f, 4d, 43, 28, 71, 19, 43, 97, d9, a6, 03}

For the decryption an arbitrary initialization vector can be used; for example:

IVD = {00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00}

Decryption using the CBC will give the following plain text. The bytes added by the padding are already removed:

PD‘ = {0d, 67, 4e, ab, 28, 16, 25, f5, 16, 7c, c6, f2, d5, 02, 69, 69}
 {fe, dc, ba, 98, 76, 54, 32, 10}

Note that the first block is different from the one in P‘.

After discarding the first block the original message is recovered.

PD = {fe, dc, ba, 98, 76, 54, 32, 10} = P

# Data encryption and licensing

## Introduction

Data Clients generally do not buy S-100 based products but are licensed to use them. Licensing is the method that Data Servers use to give Data Clients selective access to up-to-date products for a given period of time.

To operate the scheme effectively there must be a means where Data Client systems can unlock the encrypted data. To unlock the data the Data Clients system must have access to the keys that were used to encrypt the licensed data files. These keys are supplied to the Data Client, encrypted, in a permit file containing a set of permits. It is these data permits that contain the encryption keys. This method is used for file based exchange of data between the Data Client and Data Server. Other frameworks and methodologies, such as data streaming may use either variations of algorithms or different key lengths, specifying in metadata how they are defined.

To make each set of data permits exclusive the keys must be encrypted using something that is unique to the Data Clients system. OEMs assign an identifier (HW\_ID) to each of their systems and provide an encrypted copy of this, in the form of a user permit, to each Data Client. The HW\_ID is encrypted and stored in the user permit.

OEMs encrypt the HW\_ID with their own unique manufacturer key (M\_KEY) so that a HW\_ID cannot be duplicated by another manufacturer. The IHO, as the Scheme Administrator, provides the Data Servers with access to the OEM M\_KEYs and can therefore decrypt the HW\_ID stored in the user permit. Data Servers encrypt their dataset keys with the manufacturers HW\_ID when producing a set of data permits.



Figure 15-4 – High level licensing diagram based on S-101 ENC products

## Conversion of bit strings to integers

### Converting bit strings to an integer

A sequence of bits *{b1, b2, …, bn}* defines an unsigned integer *I* number by:

$I=b\_{1}2^{n-1}+b\_{2}2^{n-2}+…+b\_{n-1}2^{1}+b\_{n}; b\_{i}\in \left\{0,1\right\}$ (1a)

Or

$I=\sum\_{i=1}^{n}b\_{i}2^{n-i}$ (1b)

The bit b1 is the most significant bit and the bit bn is the least significant bit of the sequence. The integer will be in the range: $0\leq I<2^{n}$.

In most implementations the bit string will be organized as a sequence of bytes *{B0,B1,…,Bm}*, with:

$B\_{m-j}=\left\{x\_{n-8j-7},x\_{n-8j-6},…,x\_{n-8j}\right\}; ∀j\in \left\{0…m\right\}$ with $x\_{i}=\{b\_{i}; ∀i i>0 0; ∀i i\leq 0 and m=⌈\frac{n}{8}⌉$ (2)

A possible implementation of converting such a byte sequence to an integer number is given by the following pseudo code.

Input: Byte sequence *B={B0, B1,…,Bm}*

Output: non-negative integer number *I*

*Let I=0
for k from 0 to m*

*I = I \*28*

*I = I + Bk*

*Return I*

### Converting an integer number to a bit string

Formula 1a and 1b describe how a bit string is related to a corresponding (non-negative) integer number. Assuming that the bit string is organized as a sequence of bytes as defined by (2) the following algorithm shows how to transform an unsigned integer number to a bit string.

*Input: a non-negative integer number I with 0<=I<2n*

*Output: a sequence of bytes B of length* $m=\{1;I=0 ⌈\frac{n}{8}⌉;I>0 $

*Let B be an empty sequence*

 *If I = 0*

*Append the byte b=0 to B*

*Else*

*While I > 0 do*

*Let* $c = I mod 2^{8}$

*Prepend c to B*

*Let* $I = I div 2^{8}$

*While the length of B is < m*

*Prepend 0 to B*

*Return B*

Note that the division by 28 is equivalent by the bit shift operation *I* >>8

### Converting an unsigned integer number to a hexadecimal text representation

The following pseudo code shows how to convert an unsigned integer number to its hexadecimal text representation. In this text representation each digit can have 16 different values.

The integer *I* is defined as:

$I=d\_{n}16^{n-1}+d\_{n-1}16^{n-2}+…+d\_{2}16+d\_{1}$ (3)

Table 15-2 – Conversion of unsigned integer to hexadecimal text

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Digit d | Bit string | Character | ASCII Code (Hex) | ASCII Code (dec) |
| 0 | 0000 | ‘0’ | 30 | 48 |
| 1 | 0001 | ‘1’ | 31 | 49 |
| 2 | 0010 | ‘2’ | 32 | 50 |
| 3 | 0011 | ‘3’ | 33 | 51 |
| 4 | 0100 | ‘4’ | 34 | 52 |
| 5 | 0101 | ‘5’ | 35 | 53 |
| 6 | 0110 | ‘6’ | 36 | 54 |
| 7 | 0111 | ‘7’ | 37 | 55 |
| 8 | 1000 | ‘8’ | 38 | 56 |
| 9 | 1001 | ‘9’ | 39 | 57 |
| 10 | 1010 | ‘A’ | 41 | 65 |
| 11 | 1011 | ‘B’ | 42 | 66 |
| 12 | 1100 | ‘C’ | 43 | 67 |
| 13 | 1101 | ‘D’ | 44 | 68 |
| 14 | 1110 | ‘E’ | 45 | 69 |
| 15 | 1111 | ‘F’ | 46 | 70 |

The algorithm is:

*Input: An unsigned integer number I*

*Output: The hexadecimal text representation S*

*Let S be an empty sequence of characters.*

*If I = 0*

*Let S = “0”*

*Else*

*While I>0*

*Let c be the character corresponding to the value* $d=I mod 16$

*Prepend c to S*

*Let* $I = I div 16$

*Return S*

### Converting a hexadecimal text representation to an unsigned integer number

The following algorithm shows how to convert a hexadecimal text representation of an unsigned integer number to the integer number itself.

*Input: A hexadecimal text representation S of an unsigned integer number S = {s1,s2,…,sm}*

*Output: An unsigned integer number I*

*Let I = 0*

*For I = 1 to m*

*I = I\*16*

*I = I + d; where d is the digit value corresponding to the character Si*

*Return I*

## The User Permit

The user permit is created by OEMs and supplied to Data Clients as part of their system so that they can obtain the necessary access to encrypted products from Data Servers. The following section defines the composition and format of the user permit.

All Data Clients with systems capable of using data, protected in accordance with the IHO Data Protection Scheme, must have a hardware identification (HW\_ID) defined by the data client built into their end-user system. Such a HW\_ID is often implemented as a dongle or by other means ensuring a tamperproof identification for each installation.

The HW\_ID is unknown to the Data Client, but the OEM will provide a user permit that is an encrypted version of the HW\_ID and unique to the Data Client’s system. The user permit is created by taking the assigned HW\_ID and encrypting it with the manufacturer key (M\_KEY). The CRC32 algorithm is run on the encrypted HW\_ID and the result appended to it. Finally the manufacturer attaches their assigned manufacturer identifier (M\_ID) to the end of the resultant string. The M\_KEY and M\_ID values are supplied by the SA and are unique to each manufacturer providing IHO Data Protection Scheme compliant systems.

The Data Client gains access to S-100 based encrypted products by supplying their user permit to the Data Server. This enables the Data Server to issue Data Permits specific to the Data Client’s user permit. Since the user permit contains the manufacturers unique M\_ID this can be used by Data Servers to identify which M\_KEY to use to decrypt the hardware ID in the user permit. The M\_ID is the last six characters of the user permit. A list of the manufacturer M\_KEY and M\_ID values is issued and updated by the SA to all Data Servers subscribing to the scheme. This list will be updated periodically as new OEMs join the scheme.

### Definition of user permit

The user permit is 46 characters long and must be written as ASCII text with the following mandatory encoding format and field lengths:

Table 15-3 – User permit field structure

|  |  |  |
| --- | --- | --- |
| **Encrypted HW\_ID** | **Check SUM (CRC)** | **M\_ID Manufacturer ID** |
| 128 bits (32 characters) | 8 characters | 6 characters |

Any alphabetic character will be written in upper case.

Example: Encoded user permit:

**AD1DAD797C966EC9F6A55B66ED98281599B3C7B1859868**

The structure of the user permit is explained in the following sub-clauses.

#### HW\_ID Format

The HW\_ID is a 16 byte hexadecimal number defined by the OEM. Such a HW\_ID can be implemented as a dongle or by other means and must ensure a tamperproof identification of each installation.

The HW\_ID will be stored in an encrypted form in the user permit. It is encrypted using the AES algorithm with the OEM M\_KEY as the key resulting in a 128 bit value (see clause 15-6.2.4). The 128 bit encrypted HW\_ID is then represented in its ASCII form in the user permit as 32 hexadecimal digits, if necessary prepending 0’s to get the 32 required digits.

Note that the size of the HW\_ID is identical to the AES block size and does not require any padding.

Example of HW\_ID is: **40384B45B54596201114FE9904220101**

Example of encrypted HW\_ID is: **AD1DAD797C966EC9F6A55B66ED982815**

(M\_KEY=**4D5A79677065774A7343705272664F72**)

#### Check Sum (CRC) Format

The Check Sum is an 8 digit hexadecimal number. It is generated by taking the encrypted HW\_ID and converting it to a 32 character hexadecimal string. The string is then hashed using the algorithm CRC32 and the 4 bytes converted to an 8 character hexadecimal string.

The Check Sum is not encrypted and allows the integrity of the user permit to be checked.

The Check Sum in the above example is calculated from:

* Example HW\_ID: **40384B45B54596201114FE9904220101**
* Example Encrypted HW\_ID: **AD1DAD797C966EC9F6A55B66ED982815**
* CRC32 Checksum: **99B3C7B1**

#### M\_ID Format

The M\_ID is a 6-character alphanumeric code expressed as ASCII text provided by the SA to the OEM. The SA will provide all licensed manufacturers with their own unique Manufacturer Key and Identifier (M\_KEY and M\_ID) combination. The manufacturer must safeguard this information.

The SA will provide all licensed Data Servers with a full listing of all manufacturer codes as and when new manufacturers subscribe to the scheme. This information is used by the Data Server to determine which key (M\_KEY) to use to decrypt the HW\_ID in the User permit during the creation of Data Client Dataset Permits.

The M\_ID in the above example is: **859868**

### M\_KEY Format

The M\_KEY is a random 16 byte hexadecimal (128 bit) number assigned to the manufacturer and provided by the SA. The OEM uses this key to encrypt assigned HW\_ID values to generate user permits. This key is also used by the Data Server to decrypt assigned HW\_IDs. Note that the size of the M\_KEY is identical to the AES block size and does not require any padding.

Example of the M\_KEY is: **4D5A79677065774A7343705272664F72** (Hexadecimal representation)

The complete example is shown in Table15-4 below:

Table 15-4 – Complete user permit – example

|  |  |
| --- | --- |
| **Field** | **Value** |
| M\_ID | **859868** |
| M\_KEY | **4D5A79677065774A7343705272664F72** |
| HW\_ID | **40384B45B54596201114FE9904220101** |
| Encrypted HW\_ID | **AD1DAD797C966EC9F6A55B66ED982815** |
| CRC32 (Encrypted HW\_ID) | **99B3C7B1** |
| Complete User Permit | **AD1DAD797C966EC9F6A55B66ED98281599B3C7B1859868** |

## The data permit

To decrypt a data file the Data Client must have access to the encryption key (see clause 15-6.2.1) used to encrypt it. Since the encryption keys are only known to the Data Server there needs to be a means of delivering this information to Data Clients in a protected manner. This information is supplied by the Data Server to the Data Client in an encrypted form known as a permit. A file is provided to deliver the data permit and it is named PERMIT.XML (see clause 15-7.4.1). This file may contain several permits based on the product coverage required by the Data Client.

The PERMIT.XML file will be delivered either on hard media or using online services in accordance with the Data Servers operating procedures. These procedures will be made available to Data Clients when purchasing a license.

Each record within the data permit file also contains additional fields that are supplied to assist OEM systems to manage the Data Clients license and permit files from multiple Data Servers, see clause 15-7.4.2.

Data Clients can obtain a licence to access products by supplying the Data Server with their user permit (see clause 15-7.3). Data Servers can then extract the HW\_ID from the user permit, using the Data Client’s M\_KEY, and create client specific permits based on this value. The format of a permit file record is described below in clauses 15-7.4.1 to 15-7.4.4.

Since data permits are issued for a specific HW\_ID they are not transferable between installations (Data Client Systems). This method of linking the permit to the installation supports the production of generically encrypted data which can be distributed to all Data Clients subscribing to a service.

The Data Clients system decrypts the permit using the assigned HW\_ID stored by hardware or software means. The decrypted keys can then be used by the system to decrypt the licensed products. Since several Data Servers can make permit files for a specific type of product, it is the responsibility of the Data Client system to manage permit files from multiple Data Servers.

### The permit file (PERMIT.XML)

The filename will always be provided in UPPERCASE as will any alphabetic characters contained in the file. The file is completely encoded in ASCII and conforms to the S-100 XML schema for permits. OEMs should be aware that all ASCII text files generated by the Protection Scheme may contain ambiguous end-of-line markers such as CR or CRLF and should be able to deal with these.

The XML schema structure is illustrated in Figure 15-5 below.



Figure 15-5 – Structure of the permit file

The PERMIT.XML file can contain multiple sections with a corresponding XML element as follows:

Table 15-5 – PERMIT.XML elements

|  |  |
| --- | --- |
| **XML element** | **Description** |
| header | File creation date, the name of the Data Server and the format version |
| products | Permits from the Data Server for the specified product |

Note that the PERMIT.XML file can contain permits for multiple products provided by the Data Server. OEMs must ensure that their end-user software is able to merge permits from multiple data servers.

### The Permit File - Header content

The following Table defines the content and format of each section within the permit XML file.

 Table 15-6 – Contents and format of PERMIT.XML

|  |  |  |
| --- | --- | --- |
| **Content** | **XML element** | **Description** |
| File name | filename | Name of resource the permit is intended for, without pathnameFormat: Character string |
| Date | issueDate | DateXML format: xs:dateExample: <issueDate>2018-03-20Z</issueDate> |
| Provider | dataserverName | Name of Data Server who has generated the permit file. The Data Server name should be consistent and use the same organizational contact as defined in S100\_ExchangeCatalogue – contactXML format: xs:string |
| Provider identifier | dataserverIdentifier | Short identifier of data server |
| Version  | version | Version number of S-100. It will be compatible with the IHO version numbering scheme X.Y.Z. For example 4.0.0Format: Character string |
| User permit | userpermit | The user permit that the permit is intended for. This allows the client system or implementer to validate the destination. The end-user system must be capable of checking if the permit is for the designated system on a multi system bridge. Character string as defined in clause 15-7.3.1Format: Character string |

### Product sections and permit records fields

Each header element in the PERMIT.XML file is followed by a single element called “products” which contains multiple “product” records, each of which contain the actual permits for those products. This allows a single PERMIT.XML file to contain permits for multiple products all destined for an end user system. The attribute “id” for each product section contains the S-100 identifier of the Product Specification to which the permits relate; for example, <product id=”S-101”>. Permit files may contain multiple pairs of header/products elements relating to different end user systems.

### Definition of the permit record

Each product element in the PERMIT.XML file contains a sequence of “permit” elements. These elements contain the actual permits for the products identified. The Table below defines the elements contained in the permit elements with a definition of the purpose of each; fields are mandatory unless otherwise stated. Note that permits are only issued for Base datasets and the same permit is used to decrypt incremental updates (if the Product Specification implements updates).

Table 15-7 – Permit record elements

|  |  |  |
| --- | --- | --- |
| **Field** | **Purpose** | **Format** |
| filename | The file name as defined in S100\_DatasetDiscoveryMetadata – fileName. It enables Data Client systems to link the correct encryption key to the corresponding encrypted file. The pathName to the file is defined in the Exchange Set Metadata | Character string |
| editionNumber | [Optional] The edition number of the product file as defined in S100\_DatasetDiscoveryMetadata - editionNumberFor products without an edition number the permit will apply to all issued datasets | Character string |
| issueDate | [Optional] If the product does not have an edition number then the issue date may be used as an alternative identifier | xs:date |
| expiry | This is the date when the Data Clients licence expires. Systems must prevent any new editions or updates issued after this date from being installed | xs:date |
| encryptedKey (EK) | EK contains the decryption key for the specified edition of the product file | 32 character hexadecimal string representing the 128 bit encrypted key |

### Permit file signatures

Each permit file will have a digital signature created by the Data Server. The digital signature will be stored in a separate file and will reuse the name of the permit file but will have “.SIGN” appended, for example permit.sign.

The content of the signature file will be the Data Server certificate and the permit file signature and it shall be encoded in accordance with the S-100 XML Schemas. The OEM system shall authenticate the Data Server certificate before authenticating the permit file before the dataset permit keys are decrypted.

### An example PERMIT.XML file

<?xml version="1.0" encoding="UTF-8"?>

<Permit xmlns="http://www.iho.int/s100/se/5.1"

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

 xsi:schemaLocation="http://www.iho.int/s100/se/5.1 https://schemas.s100dev.net/schemas/S100/5.1.0/S100SE/20230327/Part15.xsd">

 <header>

 <issueDate>2018-03-20Z</issueDate>

 <dataServerName>Primar</dataServerName>

 <dataServerIdentifier>PR</dataServerIdentifier>

 <version>1.0.0</version>

 <userpermit>267C3AD506E69B1ED18AA5ECC7FFDE6E7C330CE8859868</userpermit>

 </header>

 <products>

 <product id="S-101">

 <datasetPermit>

 <filename>101GB40079ABCDEF.000</filename>

 <editionNumber>10</editionNumber>

 <expiry>2022-12-31</expiry>

 <encryptedKey>2E16E07E451FF1854156634DA3DD3FB8</encryptedKey>

 </datasetPermit>

 <datasetPermit>

 <filename>101NO32802411223.000</filename>

 <editionNumber>5</editionNumber>

 <expiry>2022-06-10</expiry>

 <encryptedKey>C714B5C0FBDF14BFE4B1F12E62CE5FF6</encryptedKey>

 </datasetPermit>

 </product>

 <product id="S-102">

 <datasetPermit>

 <filename>102NO329048208.h5</filename>

 <editionNumber>1</editionNumber>

 <expiry>2022-12-31</expiry>

 <encryptedKey>50BBC28B6793E1C3966B45FB2932E1BE</encryptedKey>

 </datasetPermit>

 </product>

 </products>

</Permit>

# Data authentication

This section specifies the mechanisms, structures and content required for the implementation of copy protections and/or authentication methods by S-100 Product Specifications. It defines standardized methods for the encryption of file based components of datasets as well as Feature and Portrayal Catalogues. Algorithms and methods for digital signature implementation are defined as well as the surrounding infrastructure required for key management and identity assurance within the IHO Data Protection Scheme.

## Introduction to data authentication and integrity checking

The digital signature technique in S-100 uses a standard algorithm and key exchange mechanism widely available and used. Digital signatures use asymmetric public key algorithms within a PKI-like infrastructure scheme to unbreakably bind a data file with the identity of the issuer.

The Scheme relies on asymmetric encryption[[1]](#footnote-1) of a checksum of a data file. By verifying the signature against the issuer’s public key, and also verifying the issuer’s public key against a top level identity, the user is assured of the signer’s identity. A detailed technical description of digital signatures is beyond the scope of this document and the reader is referred to the Digital Signature Standard (DSS – FIPS Publication 186) for a more detailed and accessible explanation. This Part of S-100 assumes a basic knowledge of digital signature terms and the operation of PKCS (public key cryptography standards) authentication schemes.

The IHO Data Protection Scheme can be considered to have three distinct phases:

1. A Scheme Administrator (SA) verifies the identity of a Data Server of S-100 products and provides the supplier with information to allow them to digitally sign their products.
2. A Data Server issues products signed with their identity (and their identity’s verification by the SA).
3. The subsequent verification by the Data Client of the Data Server’s identity; its association with the SA; and the integrity of the product data.

A Domain Coordinator may also act as an intermediary between the Data Server and the SA. The SA certifies the identity of the Domain Coordinator who then, in turn, can certify the identities of Data Servers they are responsible for.

It should be noted that the S-100 digital signature mechanism is not intended solely for S-100 Product Specifications’ data files. It is possible to both encrypt (and issue permits for) and digitally sign any file based data and the mechanisms described in this Part will be used to sign catalogues and other supplementary files, including Feature and Portrayal Catalogues.



Figure 15-6 – The process of data server and digital signature creation

## Data Protection Scheme setup, Data Server signup and authentication sequence

The following is a list of the steps taken by each body in the Data Protection Scheme during the digital signing of data files.

1. Scheme Creation and Setup (once only, at the instigation of the Data Protection Scheme):
	1. The SA creates their own public/private key pair and self-signs it.
	2. The SA puts their self-signed Public Key (also known as their “certificate”) in the public domain.
	3. The SA Public Key is embedded where required in OEM systems.
2. Data Server setup (once only):
	1. The Data Server creates a Public and Private Key pair.
	2. The Data Server signs their Public Key (with their Private Key) creating a Self Signed Key (also sometimes called a “certificate signing request”).
	3. The Data Server’s Self Signed Key (SSK) is sent to the SA (or Domain Coordinator) for validation when applying to join the IHO S-100 Data Protection Scheme. Any other requirements and duties within the Data Protection Scheme are issued to the prospective Data Server at this stage.
3. Data Server Identity Verification:
	1. If accepted the SA verifies the Data Server’s SSK and identity.
	2. The SA signs the Data Server’s SSK with its own Private Key to produce an SA signed Data Server Certificate.
	3. The Data Server certificate is then returned to the Data Server.
	4. The Data Server verifies that the certificate signs their Public Key against the SA Public Key.
4. The Data Server can then produce digital signatures of data files. Digital signatures of Feature and Portrayal Catalogues can also be produced by some Scheme participants as required.

## Verification of digital signatures

The verification of digital signatures by a client system takes the following steps:



Figure 15-7 – The process of data authentication by a client system

## Data Formats and standards for digital signatures, keys and certificates

The following categories of content are required for data authentication:

1. Key pairs, Private and Public Keys. These are all PEM encoded Elliptical Curve Digital Signature Algorithm (ECDSA) keys together with their elliptic curve domain parameters. The formats are described in the RFC 5915 (Private Keys) and RFC 5480 (identifier and Public Key format). The ’unrestricted’ algorithm identifier must be set to:

id-ecPublicKey OBJECT IDENTIFIER ::= {
 iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }

The parameter namedCurve must be set to:

secp384r1 OBJECT IDENTIFIER ::= {
 iso(1) identified-organization(3) certicom(132) curve(0) 34 }

1. Certificate signing requests and digitally signed Public Keys. When a Public Key is itself digitally signed it is referred to as a “certificate” (because the Public Key is “certified” by the use of the Private Key to authenticate it). When the Public Key is signed by its corresponding Private Key it is referred to as a “self-signed” certificate. These are laid out as X.509 records and can be either DER or PEM encoded to be sent to the SA for signing. When embedded within XML files keys should be PEM encoded so that the plain text can be inserted as an XML element. To conform with this standard the CSRs and the certificates should define the hash algorithm to be used for the creation of the digital signatures to SHA384 (also known as SHA2-384).
2. The digital format of the SA signed Public Keys (“certificates”) is X509v3 format encoded as PEM.

The distinguished name (DN) in the X.509 certificate forms part of the immutable content of the certificate (that is, it cannot be changed without invalidating the certificate). The roles of the scheme participants and the domains they are assigned to may be encoded in the DN. The IHOs operational procedures for the Data Protection Scheme will implement whatever specific procedures are required for the formatting of this content. The SA may place restrictions on the values allowed in the DN’s components (for example, the Common Name or the Organization) and the format of such identifiers in order to manage the operation of the Data Protection Scheme amongst its participants.

The policies and procedures implemented by the SA are not within scope of this Part of S-100 and shall be defined elsewhere. Using the DN to define the certificateRef fields can also assist implementers in selecting the correct certificate when verifying a digital signature. This may also be mandated by the SA as it specifies how the Data Protection Scheme is operated.

PEM format defines a textual encoding of the multiple large numbers required by the ECDSA algorithm (along with the ECDSA parameters required by the ECDSA algorithm). PEM encoding (originally developed for email encoding but used extensively in the encryption community for encoding of long integers used for keys and digital signatures) allows the embedding of Public Keys and Data Server certificates within XML files for permit file XML creation, the creation of catalogue and support file metadata and the production of digital signatures of Portrayal and Feature Catalogues. Digital signatures of S-100 data files must be embedded in the catalogue metadata and serve the dual purpose of a checksum against the unencrypted data file and the authentication of its source. Therefore they must be produced prior to any compression and encryption mechanism as copy protection is itself optional.

The SA Certificate represents an ECDSA Public Key provided as a PEM encoded text file. The S-100 Part 15 SA Certificate will always be available in a file called IHO.PEM. The IHO.PEM file is available from IHO at <http://www.iho.int>.

Digital Signatures in S-100 are implementations of the Digital Signature Standard (DSS). The DSS uses an approved hash function to create a message digest (hash) of the file content. The message digest is then input to the Elliptic Curve Digital Signature Algorithm (ECDSA) to generate the digital signature for the message using an asymmetric encryption algorithm and the ‘Private Key’ of the signer’s key pair. S-100 file based authentication uses the curve NIST P-384 and the hashing function SHA384. Other frameworks or data streaming via APIs may use different curves and hashing functions.

In the ECDSA algorithm a signature is a sequence of two integers. By convention these are referred to as R and S (an “R,S pair”). The format of digital signatures when embedded in XML files is as follows:

<digitalSignature id=”primar” certificateRef=”root”>
MGQCMDP17NEJXU7gzwTQAp2lgyDzJd1agCeoZ6FZOMGFRmV4sPfzAUhlC3hdj+DF3n2n/QIwPYzh15YiBgJ5Aph11kFUjLywzjDZGHYm/GyjxeCL/8FnOviMwccTlxh65fNkL0eg==</digitalSignature>

The encoding of the two R,S large integers is a Base64 ASN.1 byte sequence[[2]](#footnote-2). These are produced natively by the openssl implementation and can be generated and verified without the need to unpack the individual R and S integers. This encoding conveniently wraps the two values unambiguously into a byte array. The ASN.1 sequence representing the R,S pair is then Base64 (RFC 4648) encoded for representation in the XML digital signature elements.

The ASN.1 schema for the above example is:

SEQUENCE (2 elem)

 INTEGER (382 bit) **33F5ECD1095D4EE0CF04D0029DA58320F325DD5A8027A867A15938C185466578B0F7F30148650B785D8FE0C5DE7DA7FD**

 INTEGER (382 bit) **3D8CE1D79622060279029875D641548CBCB0CE30D9187626FC6CA3C5E08BFFC1673AF88CC1C71397187AE5F3642F47A0**

The digital signature also contains the following attributes:

1. An “id” attribute to act as an identifier.
2. A certificateRef attribute identifying the dataserver certificate with the correct public key in it for authentication. If the signature is authenticated by the SA then the certificateRef is the identifier of the SA, defined in the schemeAdministrator element of the XML container type.

These attributes are described in Clause 15-8.8.

## Creation of key material and certificate signing requests (signed Public Keys)

The commonly used “openssl package” provides a public domain, open source tool for production of key material in the required open standards specified within this Part.

Table 15-8 below shows basic command line examples for creation of the Public and Private Key pairs, certificate production and digital signing of data files.

### SA setup

This procedure is performed once only. The command SA-1 in the Table sets up a new set of ECDSA parameters and the SA-2 command creates the SA’s “root certificate” - their self-signed key which self-certifies their identity.

When a Data Server creates an X509 certificate signing request (CSR), the SA signs it using command SA-3. This creates a signed version of the Data Server’s Public Key. The PEM encoded version of the “ds.crt” file is what is embedded in both permit files and catalogue metadata as the “Data Server certificate”.

IHO will as scheme administrator issue a scheme root certificate which is available for download on the IHO web site. It will only be used to create Data Server certificates. IHO will also become a Data Server of its own protection scheme and use this certificate kes to digitally sign S-100 standard files; e.g. portrayal/feature/interoperability catalogues.

Table 15-8 – Creation of Public and Private Key pairs – basic commands

|  |  |
| --- | --- |
| **Task** | **Command** |
| SA-1 ~~create the SA Private Key~~ Create the Scheme Administrator private key using ECDSA algorithm with vector P-384 | ~~openssl ecparam -name secp384r1 -genkey -out sa-priv.pem~~ openssl ecparam -name secp384r1 -genkey -out sa-privatekey.pem |
| SA-2 ~~create the SA self-signed certificate~~ Create Scheme Administrator self-signed root key | ~~openssl req -new -x509 -key sa-priv.pem -sha384 -out sa.crt -days 365~~ openssl req -new -x509 -key sa-privatekey.pem -sha384 -out sa-root.crt -days 365 |
| SA-3 create and sign a Data Server certificate | ~~openssl x509 -req -in ds.csr -CA sa.crt -CAkey sa-priv.pem -out ds.crt -sha384 -days 365~~ openssl x509 -req -in ds.csr -CA sa-root.crt -CAkey sa-privatekey.pem -out dataserver.crt -sha384 -days 365 |

### Data Server setup

The Data Server sets up their identity with the SA by using the once only process described by commands DS-1 to DS-4. This delivers an SA signed certificate to the Data Server which is included with every delivery of signed material to the Data Client.

A Data Server must encode its Producer Code as an urn:mrn entry in the Certificate Signing Request (CSR) file which is sent to the Scheme Administrator. This information will be used by an ECDIS to determine if datasets are official or unofficial. A Data Server can request a Producer Code at the IHO Geospatial Information Registry web site.

The Producer Code information shall be encoded in the CSR Common Name (CN) field in the following format: urn:mrn:iho:org:AAAA:DDDDD.

A Data Server must encode its role in the Certificate Signing Request (CSR) file which is sent to the Scheme Administrator. Role is encoded in the certificate State or Province field (ST) in the CSR file in accordance with one of the following options:

* DATA\_PRODUCER - producing data content for live navigation under SOLAS. This data is "official"
* OTHER\_DATA\_PRODUCER - produce data content which is "unofficial"
* DATA\_AGGREGATOR - RENCs/Aggregators - validate, distribute and (sometimes) digitally sign data on behalf of their members. These organisations do not create data content but can "stamp" data as "official"
* AGGREGATOR - aggregate data together for the purposes of running a service for end users. They can only digitally sign datasets.
* SCHEME\_ADMINISTRATOR – IHO as Scheme Administrator (only for use by IHO)

Table 15-9 – Data Server setup commands

|  |  |
| --- | --- |
| **~~Task~~** | **~~Command~~** |
| ~~DS-1 create the Data Server Private Key~~ | ~~openssl ecparam -name secp384r1 -genkey -out ds-key.pem~~ |
| ~~DS-2 split Public Key from Private Key~~ | ~~openssl ec -outform pem -in ds-key.pem -out ds-public-key.pem -pubout~~ |
| ~~DS-3 create a signing request~~ | ~~openssl req -new -sha384 -out ds.csr -key ds-key.pem~~ |
| ~~DS-4 verify received certificate from SA~~ | ~~openssl verify -verbose -CAfile sa.crt ds.crt~~ |
| ~~DS-5 make data file~~ | ~~echo "hello world" > hw.txt~~ |
| ~~DS-6 sign data file~~ | ~~openssl dgst -sha384 -sign ds-key.pem -out signature.bin hw.txt~~ |
| ~~DS-7 encode signature as Base 64~~ | ~~openssl enc -base64 -in signature.bin -out signature.b64~~ |
| ~~DS-8 verify signature~~ | ~~openssl enc -d -base64 -in signature.b64 -out signature.bin~~~~openssl dgst -sha384 -verify ds-pub.pem -signature signature.bin hw.txt~~ |
| **Task** | **Command** |
| DS-1: Create the Data Server private key using ECDSA algorithm with vector P-384 | openssl ecparam -name secp384r1 -genkey -out ds-privatekey.pem |
| DS-2: Save Data Server public key associated with private key | openssl pkey -in ds-privatekey.pem -pubout -out ds-publickey.pem |
| DS-3: Create Data Server certificate signing request to be sent to IHO SA. Note requirements for encoding certificate Common Name (CN) and State or Province (ST) | openssl req -new -sha384 -out ds.csr -key ds-privatekey.pem |
| DS-4: Verify signed Data Server certificate received from SA | openssl verify -verbose -CAfile sa-root.crt dataserver.crt |
| DS-5: Create a signature for the file hw.txt | openssl dgst -sha384 -sign ds-privatekey.pem -out signature.bin hw.txt |
| DS-6: Encode the signature from DS-5 as Base64 | openssl enc -base64 -in signature.bin -out signature.b64 |
| DS-7: Extract the Data Server public key from the Data Server certificate | openssl x509 -in dataserver.crt -pubkey -noout -out ds-publickey.pem |
| DS-8: Encode the signature in digital form | openssl enc -d -base64 -in signature.b64 -out signature.bin |
| DS-9: Verify the signature of the file hw.txt  | openssl dgst -sha384 -verify ds-publickey.pem -signature signature.bin hw.txt |

The commands DS-5 to DS-8 show how a simple text file “hello world” can be created, signed with the Data Server’s private key to create an ECDSA-P384 signature, and then verified. DS-7 creates a base 64 encoded signature which can be used for embedding in an XML file (either PERMIT.XML or the catalogue metadata as required) according to the relevant Part of S-100.

**<digitalSignature>**

**MGQCMDP17NEJXU7gzwTQAp2lgyDzJd1agCeoZ6FZOMGFRmV4sPfzAUhlC3hdj+DF**

**3n2n/QIwPYzh15YiBgJ5Aph11kFUjLywzjDZGHYm/GyjxeCL/8FnOviMwccTlxh6**

**5fNkL0eg==**

**</digitalSignature>**

## Digital certificate example

Digital certificates will be PEM encoded for easy exchange and embedding in XML files. The following is an example of a PEM encoded Data Server certificate. The commands listed in the previous section format the public keys and the certificate signing request appropriately for communication between the SA and the DS. When embedding the digital certificates in XML elements, the header and footer lines are omitted.

The catalogue file of a S-100 based Exchange Set will contain a copy of all the Data Server certificates in use by all the files included in the Exchange Set with the exception of the SA root certificate which is installed separately by the end user. An identifier representing the SA root certificate is included in the exchange catalogue certificates by a “schemeAdministrator” element with an “id” attribute.

Each XML element containing a certificate will have a unique identifier attribute “id”. Each XML certificate definition will also include an attribute, “issuer” defining the id of the issuer, either the SA (identified by the schemeAdministrator id) or a domain coordinator (whose certificate will also be included in the Exchange Set). The example below shows an extract from an Exchange Set Catalogue header with the SA certificate (not included) given the id “root”. An SA signed Data Server certificate with id “DS1” is then included with the PEM encoded certificate.

 **<S100XC:certificates>**

 **<S100CE:schemeAdministrator id="root"/>**

 **<S100CE:certificate id="DS1" issuer="root">**

**MIICDjCCAZMCFEvCGmio4FLGYU9VtSiIjkR3n+i6MAoGCCqGSM49BAMDMFoxCzAJ**

**BgNVBAYTAk1DMRUwEwYDVQQHDAxEZWZhdWx0IENpdHkxHDAaBgNVBAoME0RlZmF1**

**bHQgQ29tcGFueSBMdGQxCjAIBgNVBAsMAS8xCjAIBgNVBAMMAWQwHhcNMjMxMTMw**

**MTczOTA0WhcNMjQxMTI5MTczOTA0WjB7MQswCQYDVQQGEwJNQzEWMBQGA1UECAwN**

**REFUQV9QUk9EVUNFUjEwMC4GA1UECgwnSW50ZXJuYXRpb25hbCBIeWRyb2dyYXBo**

**aWMgT3JnYW5pc2F0aW9uMSIwIAYDVQQDDBl1cm46bXJuOmlobzpvcmc6MDBBQTox**

**ODEwMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEfnOz0pGcPnvTXIYVhfvWsFm5+gf0**

**5QRlfCtfswveUijttUHrZJUDZSBf5s15tEEAaseQqDpJJcR9z354GN4uzpqHPELL**

**zNaahZ+oYBois44W4Y5Qo+NfH5iaRHbmsNOiMAoGCCqGSM49BAMDA2kAMGYCMQDc**

**lFEyN3iFINm/5O1mKp/8HwPxnDwkH7tgBnY8PBLQk69vTqPOow3cieJN44EM9rsC**

**MQCo+v/K7P1eanGRurkLOstFoEcNySgErIDFQ7sCYF8/E3/onf5/q81wMH66DBJF**

**IHU=**

 **</S100CE:certificate>**

## Creation of digital signatures by a Data Server

The Data Server creates a digital signature for the required data files using the ECDSA algorithm and their Private Key, see clause 15-8.4.

All files included in an S-100 Exchange Set must have their signatures encoded in either the S100\_DatasetDiscoveryMetaData-digitalSignature or S100\_SupportFileDiscoveryMetadata-digitalSignature elements.

The digitalSignatureReference field must be encoded “**ECDSA-384-SHA2**”.

The Data Server certificate must always be provided with a digital signature. It enables the OEM to authenticate the certificate using the SA public key and checking the certificate validity. The Data Server public key can be extracted from the certificate and used to authenticate the dataset file.

The individual id attributes can be used as a look-up by an OEM when a digital signature is defined. It reduces the need to repeat a Data Server certificate every time a signature is encoded.

The same XML elements for a Data Server certificate and digital signature defined in the Exchange Set catalogue are also used for digitally signing auxiliary files not included in the catalogue metadata; for example, catalogue and permit files. These are included in the S-100 XML Schemas and are self-contained with all necessary Data Server certificates included in them.

Since it is possible for Domain Coordinators (for example, IMO) to create Data Server certificates for participants of their domain, the following mechanism must be used to ensure the Data Client system can perform a certificate path validation:

1. The Data Server must always include the digital certificate of its Domain Coordinator to ensure the Data Client OEM has all the certificates required to perform a full certificate path validation without any external access.
2. When a Data Server certificate is defined in the catalogue metadata, it will include a data server ID and a reference to the issuer. The OEM should look up the issuer certificate and use it for Data Server authentication.
3. The OEM should verify the identity of the certificate issuer to verify the correct domain certificate to be used for Data Server certificate authentication. This is done prior to verification of the signature in accordance with ECDSA. All certificates in the Exchange Set shall be authenticated by the SA, either directly or through indirect authentication by one or more Domain Coordinators.

The digital signature is used in the catalogue metadata (and support file metadata) in two areas:

* The ECDSA digital signature of the data file, the R,S pair is embedded within the appropriate XML element according to the S-100 XML Schemas and base64 encoded; for example:

<digitalSignature id=”sig1” certificateRef="PRIMAR"> MGQCMDP17NEJXU7gzwTQAp2lgyDzJd1agCeoZ6FZOMGFRmV4sPfzAUhlC3hdj+DF3n2n/QIwPYzh15YiBgJ5Aph11kFUjLywzjDZGHYm/GyjxeCL/8FnOviMwccTlxh65fNkL0eg==
</digitalSignature>

* The Data Server certificate (which remains constant). This is encoded as per clause 15-8.4 and should be embedded in the header of the catalogue metadata. This certificate provides the Public Key against which the digital signature (and the file content) is verified. The Data Server certificate is itself signed by the Scheme Administrator (or an intermediate Domain Coordinator) and it is the responsibility of the implementer to ensure that a separately installed root certificate from the SA is available on the implementing system. Data Server certificates should be authenticated prior to authentication of the dataset file.

The Data Server certificate only needs to be included in full a single time in the Exchange Set metadata. Since the certificate does not change it can be referred to by its “id” attribute when referenced by multiple digital signatures.

Another encoding of a digital signature is the PERMIT.SIGN file which holds a standalone signature of permit file content created by the Data Server issuing the permit. The PERMIT.SIGN file is a self-contained digital signature containing elements defining the file name, the digital signature and any certificates (and intermediate Domain Coordinator certificates) required.

Data Client systems shall verify the authenticity of the permit file to ensure the signature is valid and authenticated by the SA prior to installation of any permits.

## Additional digital signatures

Additional digital signatures can be added by appending extra digital signature entries to the catalogue entry. This can express a list of certified identities signing an individual resource. This is an optional enhancement to the digitalSignature, the minimum being a single digitalSignature, verifying the content of a single resource (for example, a permit, catalogue, dataset or supplementary dataset file) against a named certificate.

* Additional digital signatures have their own XML type and can either sign the resource itself, or an existing signature of the resource.
* Additional digital signatures of the resource are appended to the exchange catalogue entry and have the same format as existing digital signatures. The dataStatus element denotes whether the signature refers to unencrypted, compressed or encrypted (and compressed) resources.
* Chains of digital signatures are implemented by use of a signatureRef attribute. A chained digitalSignature signs the content of another digital signature of the resource in the exchange catalogue. In this case the content signed is the ASN.1 byte array representing the R,S pair of the referenced signature.
* Each signature in the chain requires a valid certificateRef and an identifying “id” attribute.

These attributes are summarized in Table 15-10 below:

Table 15-10 – Additional digital signature attributes

|  |  |
| --- | --- |
| **Attribute** | **Purpose** |
| id | Unique identifier of the digital signature value |
| certificateRef | The public key which the signature can be verified against. This is only optional if the signed public key is included in a digital signature element itself, otherwise it is mandatory |
| dataStatus | [For data signatures only] whether the signature is of an unencrypted resource, one which is compressed only (such as an archive of multiple resources) or encrypted (and compressed) |

A full example, contained within a datasetDiscoveryMetadata element, follows. In this example the dataset discovery metadata specified a datafile. The first signature “s1” signs the dataset resource (no “ref” attribute is required), signature “s2” signs the encrypted data and signature “s3” signs signature s2.

[datasetDiscoveryMetadata entry]

 <S100XC:signature id="s1" certificateRef="PROD1" dataStatus="Unencrypted">

 MEUCIQCplrd+/Bb436FwXQWxgwxdcj9BhMN+EiMmZD4/6khpWwIgfLy70alp7pZSEeR27zThhQCl

OVA/ST01C+75Ond1Pu0=

 </S100XC:signature>

 <S100XC:additionalSignature>

 <S100\_SE\_SignatureOnData id="s2" certificateRef="RENC1" dataStatus="Encrypted">

MEYCIQCg+OdfUcfGJUxUKd53NmtGJ9jVOTACrKTRQM96KE0yCgIhAJ35u1aQjej7absi/V1lVOTS

DtKPUjxWAh+/DOWj+IOW
 </S100\_SE\_SignatureOnData>

 </S100XC:additionalSignature>
 <S100XC:additionalSignature>
 <S100\_SE\_SignatureOnSignature id="s3" certificateRef="DIST1" signatureRef="s2">

 MEUCIQCTGuSnqrbdQmO8ar4DdRGOjF8n5CI/9f/pGhDPeB2QhQIgMawaStd1wWJXiw1aDpz2JV/r

F9Hsx2txMN/3f2t8FIM=

 </S100\_SE\_SignatureOnSignature>
 </S100XC:additionalSignature>

## Verifying Data Integrity and Digital Identity with an S-100 digital signature

Digital signature verification is an algorithm which operates on three independent pieces of data (all formatted in line with this Part of S-100):

1. Some **content** which requires validation (the format of this content is arbitrary);
2. A **Public Key**, suitably encoded. In the ECDSA algorithm adopted this Public Key is a single number together with a set of ECDSA parameters (three numbers);
3. A **signature**. In the ECDSA algorithm a signature is composed of two numbers; by convention these are referred to as R and S (an R,S pair).

A signature verification process identifies whether the R,S pair authenticate the content against the given Public Key. This can only result in a true or false result.

ECDSA digital signature verification achieves two results:

* **Authentication**: The implementing system verifies the Data Server Public Key (“**content**”) and the signature in the Data Server certificate (“**signature**”) against the SA Public Key (or Domain Coordinator) (“**Public Key**”) to confirm that the supplier's Public Key in the certificate is valid and that the Data Server is a bona fide member of the S-100 Data Protection Scheme. If a Domain Coordinator is provided then the identity of the Domain Coordinator must also be checked against the SA Public Key.
* **Integrity Check**: The implementing system verifies the data signature (“**signature**”) and the Data Server Public Key in the Data Server certificate (“**Public Key**”) against the data file (“**content**”). This verifies the content of the data file.

If this validation check is successful then it proves that the data has not been corrupted in any way and that the identity of the Data Server within the dataset signatures is validated by the SAs identity as defined in the SA root certificate. The SA root certificate containing its public key must be installed separately on the end user system and is not packaged with the Exchange Set metadata.

## MRN specifications

In order to support discoverability of Part 17 Exchange Set resources the following MRN namespaces are defined by this Part of S-100. These are intended to be used to enable discovery of dataset supplementary resources by unique cryptographic hash or digital signature. The algorithm used to define the hash or signature is embedded in the MRN.

Tables 15-11 and 15-12 below show the specifications for digital signature and hash MRNs in S-100. All fields are mandatory and case-insensitive.

Table 15-11 – S-100 digital signature MRN

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Example** |
| Prefix | urn:mrn:iho:s100:dsig |  |
| Algorithm | From digitalSignatureReference (clause 15-8.11.8) |  ECDSA-384-SHA2 |
| Value | Computed digital Signature value | **MGUCMQCd9T4ggpAeVA/6zB0HWCXTsUOaD56lM4UitkNXrYa5rURtLwiWH2D/ZkmYRY1LTO8CMHIYHpBXvr7HwY6+W36bXnR5ylc8QTN7vc9WH/Zmo5Ck1IH02RUbS286RnYXUEP3WQ==** |
| Example | urn:mrn:iho:s100:dsig:ecdsa: MGUCMQCd9T4ggpAeVA/6zB0HWCXTsUOaD56lM4UitkNXrYa5rURtLwiWH2D/ZkmYRY1LTO8CMHIYHpBXvr7HwY6+W36bXnR5ylc8QTN7vc9WH/Zmo5Ck1IH02RUbS286RnYXUEP3WQ== |

Table 15-12 – S-100 cryptographic hash MRN

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Example** |
| Prefix | urn:mrn:iho:s100:hash |  |
| Algorithm | digitalSignatureReference (clause 15-8.11.8) | SHA-256 |
| Value | Computed cryptographic hash expressed as hexadecimal | **a948904f2f0f479b8f8197694b30184b0d2ed1c1cd2a1ec0fb85d299a192a447** |
| Example | urn:mrn:iho:s100:hash:sha256:a948904f2f0f479b8f8197694b30184b0d2ed1c1cd2a1ec0fb85d299a192a447 |

Page intentionally left blank

## Exchange catalogue metadata and standalone schema element specification

 

Figure 15-8 – Data protection – class details

### S100\_SE\_CertificateContainerType

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Role Name | Name | Description | Mult. | Data Type | Remarks |
| Class | S100\_SE\_CertificateContainerType | A set of signed public key certificates  | - | - | Used in S-100 Part 17 Exchange Catalogues |
| Attribute | schemeAdministrator | The scheme administrator identity | 0..1 | CharacterString | The identity of the Scheme Administrator is contained in the “id” attribute of the schemeAdminstrator element. The scheme Adminstrator certificate is NOT included in catalogue metadata as it is independently verified by the implementing systemThe encoding of IHO as schemeAdministrator is <S100SE:schemeAdministrator id="IHO"/> |
| Attribute | certificate | A signed public key certificate | 1..\* | Base 64 encoded Character String | Conforms to X.509 encoding. Contains a digitally signed identifier of an entity |

### StandaloneDigitalSignature

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Role Name | Name | Description | Mult. | Data Type | Remarks |
| Class | StandaloneDigitalSignature | A single digital signature | - | - | - |
| Attribute | filename | The filename of the content signed | 1 | CharacterString | The filename of the resource signed |
| Attribute | certificates | Any certificates required to authenticate the signature against the SchemeAdministrator | 1 | S100\_SE\_CertificateContainerType |  |
| Attribute | signature | A single digital signature | 1 | S100\_SE\_DigitalSignature | The signature of the file resource |

### S100\_SE\_DigitalSignature

The class S100\_SE\_DigitalSignature is realized as one of either S100\_SE\_SignatureOnData (a digital signature of a particular identified resource) or an additional digital signature defined using the class S100\_SE\_AdditionalSignature, each of which is either a S100\_SE\_SignatureOnData or S100\_SE\_SignatureOnSignature element as described in clause 15-8.8. S-100 Part 17 metadata thus allows for multiple digital signatures, a single mandatory S100\_SE\_SignatureOnData and any number of additional signatures, either of the data or other signatures.

### S100\_SE\_SignatureOnData

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Role Name | Name | Description | Mult. | Data Type | Remarks |
| Class | S100\_SE\_SignatureOnData |  | - | Base64 encoded digital signature value (clause 15-8.4) | - |
| Attribute | id | Identifier of the digital signature | 1 | CharacterString | Every signature entry has a unique identifier |
| Attribute | certificateRef | Signed Public Key | 1 | CharacterString | Identifier of the certificate against which the digital signature validates |
| Attribute | dataStatus | The digital signature | 1 | DataStatus | The digital signature value, calculated from the specified algorithm |

### S100\_SE\_SignatureOnSignature

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Role Name | Name | Description | Mult. | Data Type | Remarks |
| Class | S100\_SE\_SignatureOnSignature |  | - | Base64 encoded digital signature value (Section 15-8.4) | - |
| Attribute | id | identifier of the digital signature | 1 | CharacterString | Every signature entry has a unique identifier |
| Attribute | certificateRef | Signed Public Key | 1 | CharacterString | Identifier of the certificate against which the digital signature validates |
| Attribute | signatureref | The digital signature referenced. | 1 |  |  |

### DataStatus

|  |  |  |  |
| --- | --- | --- | --- |
| Role Name | Name | Description | Remarks |
| Enumeration | DataStatus | The state of data when a digital signature is created |  |
| Value | Unencrypted | The data is unencrypted and uncompressed | For example, supporting resources |
| Value | Encrypted | The data is compressed and encrypted | For example, copy protected datasets |
| Value | Compressed. | The data is compressed only | For example, archives of multiple resources |

### S100\_SE\_DigitalSignatureReference

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Role Name | Name | Description | Code | Remarks |
| Enumeration | S100\_SE\_ DigitalSignatureReference | A reference to a cryptographic algorithm used in an implementation of Part 15 |  | Only ECDSA is currently used in implementations of S-100 for file based transfer of data to ECDIS. Other values are included for interoperability with other implementations by external standards. See clause 15-8.4 |
| Value | RSA |  | 1 | RSA with key length >= 2048 bits |
| Value | DSA |  | 2 | DSA with key length >= 2048 bits |
| Value | ECDSA |  | 3 | ECDSA with key length >= 224 bits. |
| Value | ECDSA-224-SHA2-224 |  | 4 | 224 bits ECDSA with SHA2-224 hashing |
| Value | ECDSA-224-SHA3-224 |  | 5 | 224 bits ECDSA with SHA3-224 hashing |
| Value | ECDSA-256-SHA2-256 |  | 6 | 256 bits ECDSA: SHA2-256 |
| Value | ECDSA-256-SHA3-256 |  | 7 | 256 bits ECDSA: SHA3-256 |
| **Value** | **ECDSA-384-SHA2** |  | **8** | **384 bits ECDSA: SHA2-384** |
| Value | ECDSA-384-SHA3 |  | 9 | 384 bits ECDSA: SHA3-384 |
| Value | AES-128 |  | 10 | AES 128 bit keys |
| Value | AES-192 |  | 11 | AES 192 bit keys |
| Value | AES-256 |  | 12 | AES 256 bit keys |

# Glossary of S-100 Data Protection Scheme and computing terms

For a list of general abbreviations used throughout S-100, see Part 0, clause 0-2. For a list of general terms and definitions used throughout S-100, see Annex A.

Table 15-13 – S-100 Data Protection Scheme terms

|  |  |
| --- | --- |
| AES  | Advanced Encryption Standard, encryption algorithm used in the scheme |
| Data Permit  | File containing encrypted product keys required to decrypt the licensed products. It is created specifically for a particular user |
| Data Client  | Term used to represent an information consumer receiving the encrypted information. The Data Client will be using a software application (for example ECDIS) to perform many of the operations detailed within the scheme. |
| Data Server  | Term used to represent an organization producing encrypted data files or issuing Dataset Permits to end-users |
| M\_ID  | The unique identifier assigned by the SA to each manufacture. Data Servers use this to identify which M\_KEY to use when decrypting the Userpermit |
| M\_KEY  | System manufacturer’s unique identification key provided by the Scheme Administrator to the OEM. It is used by OEMs to encrypt the HW\_ID when creating a userpermit |
| HW\_ID  | The identifier assigned by an OEM to each implementation of their system. This value is encrypted using the OEM’s unique M\_KEY and supplied to the data client as a userpermit. This method allows data clients to purchase licences to decrypt datasets |
| PKCS | Public Key Cryptography Standards |
| IV | Initialization Vector used by the AES-CBC encryption algorithm |
| SA  | Scheme Administrator. IHO is responsible for maintaining and coordinating all operational aspects and documentation of the Protection Scheme |
| SHA | Secure Hash Algorithm  |
| SSK  | Self Signed Key (Self Signed Certificate File)  |
| User Permit  | Encrypted form of HW-ID uniquely identifying the Data Client system  |

Table 15-14 – Computing terms

|  |  |
| --- | --- |
| CRC  | Cyclic Redundancy Check  |
| XOR  | Exclusive OR |

Page intentionally left blank

1. Asymmetric cryptography relies on algorithms where encryption and decryption take place with different cryptographic keys. Therefore one person can encrypt data and make available a decryption key for others to decrypt it. These keys are referred to as the “private key” and the “public key”, collectively known as a “key pair”. [↑](#footnote-ref-1)
2. Abstract Syntax Notation One (ASN.1) is a standard interface description language for defining data structures that can be serialized and deserialized in a cross-platform way. It is broadly used in telecommunications and computer networking, and especially in cryptography. <https://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One> [↑](#footnote-ref-2)