**S-100 – Part 15**

**Data Protection Scheme**

Page intentionally left blank

**Contents**

Preface 1

15-1 Scope 3

15-2 Normative References 3

15-3 General Description 3

15-4 Participants in the Protection Scheme 4

15-4.1 Scheme Administrator 4

15-4.2 Data Servers 4

15-4.3 Data Clients 5

15-4.4 Original Equipment Manufacturers 5

15-4.5 Participant Relationships 5

15-4.5.1 Domain Coordinator 5

15-5 Data compression 6

15-5.1 Overview 6

15-5.2 Compression Algorithm 6

15-5.3 Encoding 6

15-6 Data encryption 7

15-6.1 What Data is encrypted? 7

15-6.2 How is it encrypted? 7

15-6.2.1 Encryption Algorithm 7

15-6.2.2 AES examples 9

15-7 Data encryption and licensing 10

15-7.1 Introduction 10

15-7.2 Conversion of bit strings to integers 11

15-7.2.1 Converting bit strings to an integer 11

15-7.2.2 Converting an integer number to a bit string 11

15-7.2.3 Converting an unsigned integer number to a hexadecimal text representation 12

15-7.2.4 Converting a hexadecimal text representation to an unsigned integer number 13

15-7.3 The User Permit 13

15-7.3.1 Definition of user permit 13

15-7.3.2 M\_KEY Format 14

15-7.4 The Data Permit 14

15-7.4.1 The Permit File (PERMIT.XML) 15

15-7.4.2 The Permit File - Header content 15

15-7.4.3 Product sections and Permit Records Fields 15

15-7.4.4 Definition of the Permit Record 16

15-7.4.5 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 Data Formats and standards for digital signatures, keys and certificates 19

15-8.4 Creation of key material and certificate signing requests (signed Public Keys) 20

15-8.4.1 SA setup 20

15-8.4.2 Data Server setup 20

15-8.5 Example Public Key 21

15-8.6 Creation of digital signatures by a Data Server 22

15-8.7 Verifying Data Integrity and Digital Identity with an S-100 digital signature 22

15-9 Glossary of S-100 Data Protection Scheme and computing terms 23

Appendix 15-A Encryption Examples (informative) 25

15-A-1 Converting bit string to an integer number 25

15-A-2 Converting an integer number to a bit string 25

15-A-3 Converting an unsigned integer number to a hexadecimal text representation 26

15-A-4 Converting a hexadecimal text representation to an unsigned integer number 27

15-A-5 Details on the encryption algorithm 27

15-A-6 Examples on AES 29

15-A-7 Diagrams on HW\_ID encryption 30

15-A-8 Diagrams on Data key encryption 31

15-A-9 Example of a user permit 32

15-A-10 Example of an encrypted data key 32

15-A-11 Example of a Data Permit 32

15-A-12 Example of HW\_ID 32

15-A-13 Example of Permit File (XML) 32

# Preface

Copyright infringement and data piracy are pervasive problems of the digital era. Electronic Navigational Charts (ENC) or other digital spatial products are not exempt from these issues. As well as the economic impact, the unofficial distribution of nautical information also gives rise to significant safety concerns. As a result, the publishers of official nautical information have sought to protect their data and provide the mariner with a certificate of authenticity through the adoption of a security scheme.

In September 2000, IHO Member States were polled on their views on developing a single IHO Recommended Security Scheme (RSS) (see: IHB Circular Letter 38/2000). Responses indicated that a large majority of the Member States wished to have their ENC data encrypted and agreed that the IHO should adopt a single RSS (see: IHB CL 15/2001 Rev.1). A majority of the Member States responding also supported the adoption of the PRIMAR Security Scheme as the IHO RSS, as it was at the time the de facto standard for ENC protection and the majority of ECDIS manufacturers had already developed the necessary decryption facilities in their systems.

The IHO Committee on Hydrographic Requirements for Information Systems (CHRIS, now HSSC: Hydrographic Services and Standards Committee), at its 13th meeting (Athens, Greece, September 2001), revisited the issue of an RSS and agreed that a small advisory expert group investigate the implications of IHB becoming the security scheme administrator for an RSS and assuming responsibility for the maintenance of an RSS.

The IHO Data Protection Scheme Working Group (DPSWG) reported back to the IHB in January 2002 that there were no technical implications to the IHB becoming the security scheme administrator and that the level of effort to administer the security scheme would be limited and within the IHB resources. The DPSWG further provided a plan to develop an IHO RSS Version 1, based on the Primar Security Scheme. This Report was endorsed by CHRIS Members in February 2002 and the DPSWG was tasked to develop Version 1 of an IHO RSS.

The results were presented to CHRIS, at its 14th meeting (Shanghai, China, August 2002), which recommended that the ENC Security Scheme, as developed by the DPSWG, be submitted to IHO Member States for adoption as an IHO RSS, and that the role as Security Scheme Administrator be transferred to the IHB. These proposals (see: IHB CL 44/2002) were approved by a majority of Member States (see: IHB CL 66/2002). As a result, Edition 1.0 of the IHO Data Protection Scheme was adopted in October 2003 as Publication S-63.

The 18th CHRIS meeting (Cairns, Australia, September 2006) tasked the DPSWG to develop a revised edition of S-63 with the following guidance:

* + There would be no introduction of new features; changes would be kept to a minimum;
* Published S-63 guidelines would be included in the standard;
* S-63 would be reorganized to group issues relevant to the IHB as Scheme Administrator, to Data Servers, and to OEMs, respectively;
* There would be a more precise description of the correct implementation of the IHO standard.

Accordingly, a draft Edition 1.1 of S-63 was prepared by DPSWG and endorsed by CHRIS at its 19th meeting (Rotterdam, Netherlands, November 2007). This was subsequently endorsed by Member States and adopted in March 2008. Edition 1.1 included supporting documentation, test data and a method to supply ENCs using “Large Media Support”.

In April 2012, small changes were made to Edition 1.1 to remove the hexadecimal limitation of M\_ID in order to extend the number of possible M\_ID values that the scheme is able to accommodate. This resulted in the publication of edition 1.1.1 of S-63.

In November 2014 an additional Annex was added to Edition 1.1.1 to provide a normative reference for the ENC update status report. This report reflects functionality required by edition 4 of the ECDIS Type Approval standard IEC61174, section 4.4.2 and Annex L. No other substantive changes were made to S-63 as a result of this additional Annex; only clarifications for users of the Data Protection Scheme on how the report is formatted and the definitions of its various fields. This resulted in this edition 1.2.0 of S-63.

The development of S-100 and the IMO resolution MSC.428(98) has required a revision of the Data Protection Scheme. The operational principles for using the Protection Scheme have been maintained, but changes have been introduced for the individual security constructs to reflect operational experience with the current version of the Protection Scheme and better harmonization with international security constructs. S-100 part 15 supports more product specifications based on the S-100 data model and where other organizations than IHO can operate as domain owners. S-100 part 15 edition 1.0.0 has selected to use international or industry standards for encryption and digital signatures, and this together with S-100 have required a change in how the information is encoded and distributed.

Changes to this Standard, as well as any further developments, will continue to be coordinated by projects team within the S-100 Working Group under HSSC Guidance.

# 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)>

IHO S-57, *IHO Transfer Standard for Digital Hydrographic Data*

ISO/IEC 13239:2002, *CRC32 checksum algorithm. Information technology -- Telecommunications and information exchange between systems -- High-level data link control (HDLC) procedures*

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

ISO/IEC 21320-1, *Document Container File – Part 1: Core*

*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](https://tools.ietf.org/html/rfc5652" \l "section-6.3)>

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 has 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 the digital products before supplying it to the Data Client. The encrypted products are then decrypted by the end-user system (for example ECDIS/ECS) prior to being reformatted and imported into the System Internal Format (for example SENC). Authentication is provided by means of digital signatures applied to the product files.

The security scheme does not specifically address how the product information can be 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 (for example DVD) and can 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 cell keys. This license is created using a unique hardware identifier of the target system and is unique to each Data Client. Consequently licenses cannot be exchanged between individual Data Clients.

The scheme uses a 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 decryption on the data client system (normally an ECDIS/ECS).

# 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.

A more detailed explanation of these terms is given below.

## 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 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 organisations 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 is the only body that can certify the identity of the other participants of the scheme.

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 S-100 Part 15.

## Data Servers

Data Servers (DS) are responsible for the encryption and digital signing of the datasets in compliance with the procedures and processes defined in the scheme. Data Servers 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 HW\_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 unique 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 unique hardware identifier (HW\_ID).

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, and the underlying product file consistency controls available in the underlying S-100 based product files.

## 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 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.

### Domain Coordinator

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 about the Data Server’s identity and contact details. The SA will distribute M\_ID and M\_KEY information directly to all Data Servers participating in the Protection Scheme when they join the scheme and more Data Clients are added.

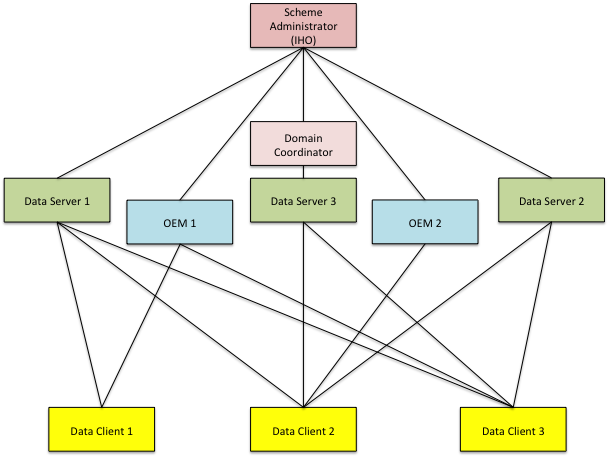


Figure 15-1 – Relationship between Protection Scheme participants

# Data compression

## 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 co-ordinate 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 if compression is being used.

## 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. 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**
* S-100\_ExchangeCatalogue-algorithmMethod with value **S100p15e1.0.0**

# 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.

The AES algorithm can only encrypt one block of plain text. For larger messages a block cipher mode of operation has to 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.

The padding methods that will be used is described in PKCS#7 [12]. 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 xx  xx | xx xx xx xx xx xx xx xx  xx **07 07 07 07 07 07 07** |
| xx xx xx xx xx xx xx xx  xx xx | xx xx xx xx xx xx xx xx  xx xx **06 06 06 06 06 06** |
| xx xx xx xx xx xx xx xx  xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx **05 05 05 05 05** |
| xx xx xx xx xx xx xx xx  xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx **04 04 04 04** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx **03 03 03** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx xx **02 02** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx **01** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx 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

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:

(3a)

(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.

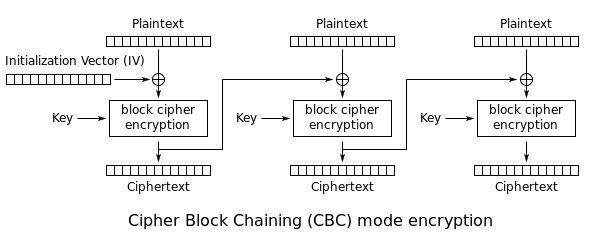


Figure 15-2 – Cipher Block Chaining (CBC) mode encryption (Source: Wikipedia)

Decryption is defined as:

(4a)

(4b)

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



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

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 recognised 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 within the data permit. 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.

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 a unique 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 cell keys with the manufacturers HW\_ID when producing a set of data permits. This makes them unique to the Data Client and as such not transferable between Data Client systems.

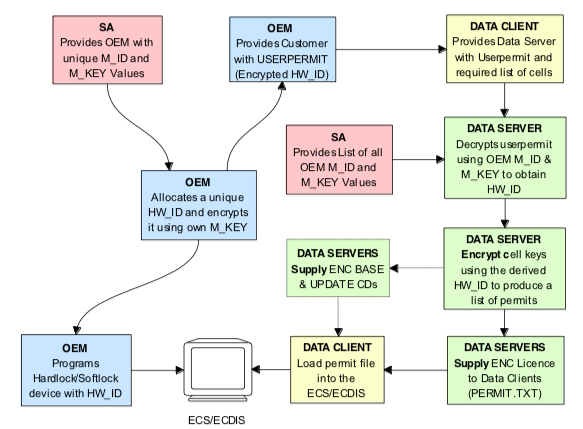


Figure 15-4 – High level licensing diagram based on S-101 ENC products

## 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:

(1a)

Or

(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: .

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

with and (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*

### 

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*

*Let B be an empty sequence*

*If I = 0*

*Append the byte b=0 to B*

*Else*

*While I > 0 do*

*Let*

*Prepend c to B*

*Let*

*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:

(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*

*Prepend c to S*

*Let*

*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 unique 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 unique and 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 28 characters long and must be written as ASCII text with the following mandatory format and field lengths:

Table 15-3 – User permit field structure

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

Any alphabetic character will be written in upper case.

Example: User permit structure:

**AD1DAD797C966EC9F6A55B66ED98281599B3C7B1859868**

The structure of this user permit is explained in the next section.

#### HW\_ID Format

The HW\_ID is a 32 digit hexadecimal number defined by the OEM. Such a HW\_ID can be implemented as a dongle or by other means ensuring 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 M\_KEY as the key resulting in a 128 bit value encoded as a 32 digit (16 bytes) hexadecimal number. The encrypted HW\_ID is then represented in its ASCII form in the user permit as 32 digit. 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: 40384B45B54596201114FE99042201

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. It 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:

* Example HW\_ID: 40384B45B54596201114FE99042201
* Example Encrypted HW\_ID: AD1DAD797C966EC9F6A55B66ED982815
* Checksum: 99B3C7B1

#### M\_ID Format

The M\_ID is a 6-character alphanumeric code expressed as ASCII representation provided by the SA. 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 cell permits.

The M\_ID in the above example is: 859868

### 

The M\_KEY is a random 32 digit hexadecimal (128 bit) number assigned to the manufacturer and provided by the SA. The OEM uses this key to encrypt assigned HW\_ID when generating user permits. This key is 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

## The Data Permit

To decrypt a data file the Data Client must have access to the encryption key (see section 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 unique 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. 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 PERMIT.XML file can contain multiple sections with a corresponding XML element as follows:

Table 15-4 – PERMIT.XML elements

|  |  |
| --- | --- |
| **XML element** | **Description** |
| header | This includes the file creation date, the name of the Data Server and the format version |
| products | Permits from the Data Server for the specified product |
| digitalSignature | The Data Server digital signature of the permit appended to the PERMIT.XML file |

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 15-5 – Contents and format of PERMIT.XML

|  |  |  |
| --- | --- | --- |
| **Content** | **XML element** | **Description** |
| Date and time | date | The field name, date and time is separated by a space character (SP <h20>). The date will be provided as YYYYMMDD and the time as HH:MM using the 24 hour clock  Example: :DATE 20180320 17:11 |
| Provider | dataserver | 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 – contact |
| Version | version | Version number of S-100. It will be compatible with the IHO version numbering scheme X.Y.Z. For example 4.0.0 |
| User Permit | userpermit | The user permit that the permit is 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 |



### Product sections and Permit Records Fields

The 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 a single end user system.

### 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.

Table 15-6 – 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 | Character string |
| Edition | The edition number of the product file as defined in S100\_DatasetDiscoveryMetaData - editionNumber | Character string |
| 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 | YYYYMMDD  (ISO-8601) |
| Encrypted Key (EK) | EK contains the decryption key for the specified edition of the product file | 32 digit hexadecimal number |

### An example permit.xml file

<permit>

<header>

<date>20180607 14:11:59</date>

<dataserver>primar</dataserver>

<version>1.0.0</version>

<userpermit>80352805938502220302542</userpermit>

</header>

<products>

<product id="S-101">

<permit>

<filename>101GB40079ABCDEF.000</filename>

<edition>10</edition>

<expiry>20183112</expiry>

<encryptedKey>2011AA840D5C2204</encryptedKey>

</permit>

<permit>

<filename>101NO32802411223.001</filename>

<edition>5</edition>

<expiry>20180610</expiry>

<encryptedKey>2065AF8E5D5C1411</encryptedKey>

</permit>

</product>

<product id="S-102">

<permit>

<filename>102NO329048208</filename>

<edition>1</edition>

<expiry>20183112</expiry>

<encryptedKey>3176BD8F5D6C0608</encryptedKey>

</permit>

</product>

</products>

<digitalSignature>

<signedpublicKey id="primar" rootKey="IHO">

</signedpublicKey>

<signature>

<R>28F549549614ED4896BECBB056BE0F36ECA172EC</R>

<S>399A5F5FC5B4DC52F1B750233F85AE3849227603</S>

</signature>

</digitalSignature>

</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-2) 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 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.

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 it is envisaged that the mechanisms described in this Part can be used to sign both Feature and Portrayal Catalogues. This will be valid for Feature and Portrayal Catalogues issued by the IHO.

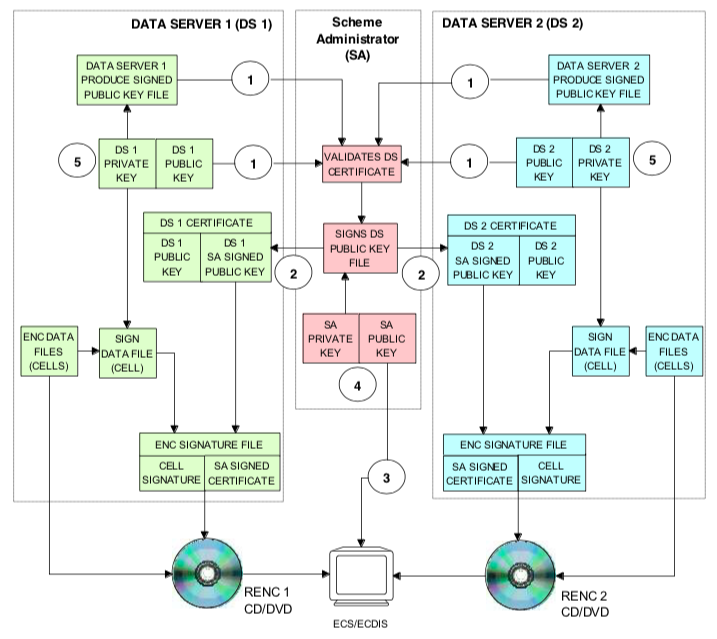


Figure 15-5 – Example of authentication process using ENC products

## 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 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 scheme participants as required.

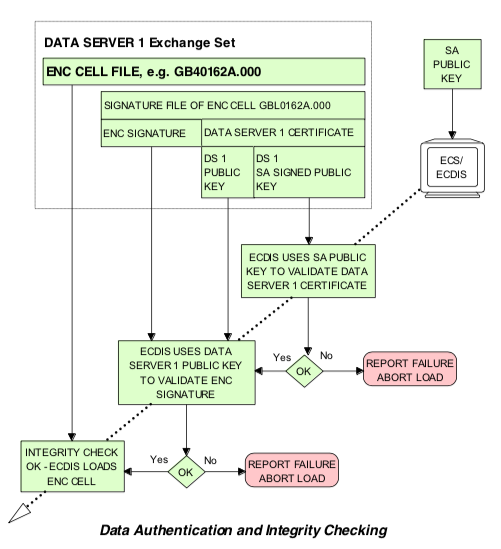


Figure 15-6 – Data authentication and integrity checking

## 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 DSA keys together with their DSA key parameters. These keys should all be 1024 bits long.
2. 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.
3. The digital format of the SA signed) Public Keys (“certificates”) is X509v3 format encoded as PEM.

PEM format defines a textual encoding of the multiple large numbers required by the DSA algorithm (along with the DSA parameters required by the DSA 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 encryption mechanism as copy protection is itself optional.

The SA Certificate represents a DSA Public Key of length 1024 bits provided, as stated, as a PEM encoded text file. The 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 the Secure Hash Algorithm (SHA256) to create a message digest (hash) of the file content that are 256 bits long. The message digest is then input to the Digital Signature Algorithm (DSA) to generate the digital signature for the message using an asymmetric encryption algorithm and the ‘Private Key’ of the signer’s key pair. The DSA key length is 1024 bits.

In the DSA 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>

302C021433796C6647CC1C55A67DC72FA7C6E157A6594B2B02145D3768B44F3A6ABA11A77178B738AD3B6A0DE344

</digitalSignature>

The R,S pair are represented by its hexadecimal encoding (digits 0-9, letters A-F).

## 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 15-7 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 DSA 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 SHA256 signed version of the Data Server’s Public Key. The PEM encoded version of the “signedicds.crt” file is what is embedded in both permit files and catalogue metadata as the “Data Server certificate”.

Table 15-7 – Creation of Public and Private Key pairs – basic commands

|  |  |
| --- | --- |
| **Task** | **Command** |
| SA-1 create DSA parameters | openssl dsaparam 1024 -out dsaparam.txt |
| SA-2 create SA root key and self signed root certificate | openssl req -x509 -sha256 -nodes -days 365 -newkey dsa:dsaparam.txt -keyout iho.key -out iho.crt |
| SA-3 sign a verified certificate signing request | openssl x509 -req -in CSR.csr -sha256 -CA iho.crt -CAkey iho.key -CAcreateserial -out signedicds.crt |

### 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-5. This delivers an SA signed certificate to the Data Server which is included with every delivery of signed material to the Data Client.

Table 15-8 – Data Server setup –commands

|  |  |
| --- | --- |
| **Task** | **Command** |
| DS-1 Create DSA parameter file | openssl dsaparam 1024 -out ICDSparam |
| DS-2 create a Data Server key  DS-3 Split Public Key from Private Key | openssl req -out CSR.csr -new -newkey dsa:ICDSparam.txt -nodes -keyout icds.key  openssl dsa -outform pem -in icds.txt -out icdspubkey.txt -pubout |
| DS-4 Create a certificate signing request | openssl req -out CSR.csr -key icds.key -new |
| DS-5 Verify received certificate from SA | openssl verify -verbose -CAfile iho.crt signedicds.crt |
| DS-6 Make data file  DS-7 Sign data file  DS-8 Create a hexadecimal signature  DS-9 Verify binary signature | echo "hello world" > hw.txt  openssl dgst -sha256 -sign icds.key hw.txt > hw.sig  openssl dgst -sha256 -hex -sign icds.key hw.txt  openssl dgst -sha256 -verify pubkey.txt -keyform pem –signature hw.sig hw.txt |

The commands DS-6 to DS-9 show how a simple text file “hello world” can be created, signed with the Data Server’s private key to create a DSA-SHA256 signature, and then verified. DS-8 creates a hexadecimal format signature which can be translated into the following XML for embedding in an XML file (either PERMIT.XML or the catalogue metadata as required.

<digitalSignature>

302C021433796C6647CC1C55A67DC72FA7C6E157A6594B2B02145D3768B44F3A6ABA11A77178B738AD3B6A0DE344

</digitalSignature>

## Example Public Key

The following is an example of a PEM encoded public key.

-----BEGIN PUBLIC KEY-----

MIIDSDCCAjoGByqGSM44BAEwggItAoIBAQD9Pm/tjwRDRMYc1FzABkQqXKpTptvQ

9EVDdl8VJSCC82hdyJQDeS1DyLCp9LNTfdp+2lkMAcSUSzBJdRUQMvww78/L/zyH

D/owQKlbvyYwUfcAfJ1LgA/5cFzL174H/XRpDuWlCKRoq959QhVW6wY5PMKHAGpx

gpzb5SiqukxqWw07XllcQqPnvIdO1OeeCTOYD7WIPS1HXwCkcP9Bcd4dfVolfDvP

azsDavtZ48qcxU53XS+W3M646qbpueFLQ66kQ1Lt0XEopJeWnxjJISGomN1vLhFx

eY0uszEwBXoG8q6T/Cf8WNBnAfj4uq1/vAiwNTNeANnDcNPtu9mlK5nxAiEAtcfr

VKQyMjfcJUpl1NeGX/qYnzXmABiAMBjqgRS5mBkCggEBAKCTVhDlBm6jADkYXmxv

HjT9ry33zNJbQIAvycSUdIw8NYFVHSDqR8hILVf9LYzbrhENu0ffHdxgImA0GZJl

duxVoxhdMHOsdKGQOlVnzv7RB961S3F4Ho46r7MVUb6z7F6JZ7oFeWt5XSlYUlbr

ecG9cXi5vfDC/HT5sR4353SkudnYaRLdcDbpc0aHqVD6DyaqockyAMXDzTHEjlK9

Lw2+mWeKqIzX7SoBfb1N0DU0Ot6R5Ni0TdL0q59rUosu7UbvIFmOd3QQxGYk0Ro/

M+9drVaEAK1zJfIvVjKuLQQPDGMhMfOktXLWi3D6UXPfRBdJSEn8PjhkmrKUNeCo

+RoDggEGAAKCAQEAkfEyvn8ALb3hAnWOmikUgjuwTxMq58/aswh4LXaIaG3UtpkL

SjnO31VH/3NG31ywAatJsmraGUijiIq4JR1m8DTI8P3lxeHcqB3ln1XYLYUw3pp3

8ABbGjuNJ4vTP+lwgOg9DPqpKsmJEb6cMtcFf7qSpMy9Hx76SO6z46r0xdMwoOkN

bHr3JNKxu9gLQZ3MY8AT7nhMcQRraO38KVahSadp35zDshlLHEd8HcCetrj1AFnN

m2AxTXeNzaLqAM6INlHZXHXO5UTu1EZW4ES4F7hdp6NwQV5ijm2IFeZg/KsuOiCW

ISLCa5sU9zw9MLrHBOF1ZqyUdBXkn4naNCZg5Q==

-----END PUBLIC KEY-----

## Creation of digital signatures by a Data Server

The Data Server creates a digital signature for the required data files using the DSA algorithm and their Private Key, see clause 15-8.3.

All files included in an S-100 exchange set must have their signatures encoded in either the S100\_DatasetDiscoveryMetaData-digitalSignatureValue or S100\_SupportFileDiscoveryMetadata-digitalSignatureValue.

The digitalSignatureReference field must be encoded “**S100p15e4.0.0”**.

The digitalSignature field must be encoded **1** (true).

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

* The DSA-SHA256 digital signature of the data file, the R,S pair is embedded within the appropriate XML element according to the following XML snippet:

<digitalSignature>

302C021433796C6647CC1C55A67DC72FA7C6E157A6594B2B02145D3768B44F3A6ABA11A77178B738AD3B6A0DE344

</digitalSignature>

* Their Data Server certificate (which remains constant). This is encoded as per clause 15-8.3 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.

Another example encoding of a digital signature is within the PERMIT.XML file which holds a signature of the entire permit file content created by the Data Server issuing the permit.

<digitalSignatureValue>

<signedpublicKey id="primar" rootKey="IHO">MIIBtjCCASsGByqGSM44BAEwggEeAoGBA  


</signedpublicKey>

<digitalSignature>

302C021433796C6647CC1C55A67DC72FA7C6E157A6594B2B02145D3768B44F3A6ABA11A77178B738AD3B6A0DE344</digitalSignature>

</digitalSignatureValue>

As can be seen from the XML taken from the PERMIT.XML the signedPublicKey represents the Data Server certificate and the <digitalSignature> element contains the R,S pair which define the signature. Data Client systems shall only verify the authenticity of the permit file using the header and product elements found in the PERMIT.XML file.

## 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;
2. A **Public Key**, suitably encoded. In the DSA algorithm adopted this Public Key is composed of a set of DSA parameters together with a Public Key;
3. A **signature**. In the DSA 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.

DSA 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 (“**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.
* **Integrity Check**: The implementing system verifies the data file 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 file has not been corrupted in any way and that the identity of the Data Server within the cell signatures is validated by the SA’s identity as defined in the SA root certificate.

# 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 15-9 – 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 end-user receiving the encrypted ENC information. The Data Client will be using a software application (for example ECDIS) to perform many of the operations detailed within the scheme. Typically, an ECDIS user | |
| Data Server | Term used to represent an organization producing encrypted data files or issuing Cell 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 | ECDIS 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 unique 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 ENC cells | |
| PKCS | Public Key Cryptography Standards | |
| IV | Initialization Vector used by the AES-CBC encryption algorithm | |
| SA | Scheme Administrator. IHO 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 15-10 – Computing terms







|  |  |
| --- | --- |
| CRC | Cyclic Redundancy Check |
| Dongle | Sometimes referred to as a hard lock device, It is a hardware device supplied by the OEMs that has the unique system identifier (HW\_ID) stored security within |
| XOR | Exclusive OR |

# Appendix 15-A Encryption Examples (informative)

## Converting bit string to an integer number

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

(1a)

or

(1b)

The bit *b*1 is the most significant bit and the bit *b*n is the least significant bit of the sequence. The integer will be in the range: .

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

*Let B be an empty sequence*

*If I = 0*

*Append the byte b=0 to B*

*Else*

*While I > 0 do*

*Let*

*Prepend c to B*

*Let*

*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:

(3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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*

*Prepend c to S*

*Let*

*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*

## Details on the 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.

The AES algorithm can only encrypt one block of plain text. For larger messages a block cipher mode of operation has to 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.

The padding methods that will be used is described in PKCS#7 [12]. 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.

|  |  |
| --- | --- |
| **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 xx  xx | xx xx xx xx xx xx xx xx  xx **07 07 07 07 07 07 07** |
| xx xx xx xx xx xx xx xx  xx xx | xx xx xx xx xx xx xx xx  xx xx **06 06 06 06 06 06** |
| xx xx xx xx xx xx xx xx  xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx **05 05 05 05 05** |
| xx xx xx xx xx xx xx xx  xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx **04 04 04 04** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx **03 03 03** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx xx **02 02** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx **01** |
| xx xx xx xx xx xx xx xx  xx xx xx xx xx xx xx xx | xx xx xx xx xx xx xx xx  xx 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

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:

(3a)

(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.

Decryption is defined as:

(4a)

(4b)

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

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 recognised from the formulas; 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 within the data permit. 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).

## Examples on AES

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 e.g.

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

## Diagrams on HW\_ID encryption

HW\_ID (128 bit)

**AES-128 encryption**

Encrypted HW\_ID (128 bit)

M\_KEY (128 bit)

HW\_ID (128 bit)

**AES-128 decryption**

Encrypted HW\_ID (128 bit)

M\_KEY (128 bit)

## Diagrams on Data key encryption

Data\_Key (128 bit)

**AES-128 encryption**

Encrypted Data\_Key (128 bit)

HW\_ID (128 bit)

Data\_Key (128 bit)

**AES-128 decryption**

Encrypted Data\_Key (128 bit)

HW\_ID (128 bit)

## Example of a user permit

HW\_ID : 123456789ABCDEF0123456789ABCDEF0

M\_KEY : 112233445566778899AABBCCDDEEFF00

M\_ID : ABXY

Encrypted HW\_ID: 4C329B7E79819AEE47E0C7AB79412EFF

CRC32 Check Sum: 19CB1B5C

User Permit: 4C329B7E79819AEE47E0C7AB79412EFF**19CB1B5C**ABXY

## Example of an encrypted data key

HW\_ID : 123456789ABCDEF0123456789ABCDEF0

Data Key : FEDCBA9876543210FEDCBA9876543210

Encrypted Data Key: CE39C3D515539299F407DC66200B3E1D

## Example of a Data Permit

TBD

## Example of HW\_ID

TBD

## Example of Permit File (XML)

TBD

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-2)