GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology
Open Firmware Loader for Tamper Resistant Secure Hardware
Version 1.3.0.14

Public Review

April 2021

Document Reference: GPC_SPE_134
(Formerly GPC_FST_134)

Contents

Figures

Tables

1 Introduction

This document specifies a security protocol based on ECC technology, the Image Format, and associated protocols.

These protocols support an Open Firmware Loader that uses a Tamper Resistant Secure Hardware (TRSH) as its Secure Hardware Platform. This Open Firmware Loader is independent of use cases supported by a Firmware.

The Open Firmware Loader associated with the Secure Hardware Platform is intended to:

- Enable the loading of a Firmware into a TRSH.
- Solve the problem of distributing a generic and blank (no Operating System) TRSH by allowing the loading of Firmware from various Firmware Makers after the issuance of the TRSH.
- Solve the logistic issues related to the distribution of TRSH in fragmented markets when addressing small volumes.
- Allow the distribution of Firmwares after TRSH issuance in order to address additional use cases.
- Solve the logistic issues related to loading Firmware containing diversified data into TRSH during the manufacturing of a device hosting the TRSH.
- Offer a generic loader, shared between multiple TRSH Makers and allowing Firmware Makers to load Firmware in an interoperable way.
- Offer confidentiality between Firmware Makers.

The Firmware to be loaded into a TRSH is distributed through containers called Images. Images are encrypted, and the Firmware inside is partitioned into Firmware Segments.

1.1 Audience

This document is intended primarily for the use of TRSH Makers, Image Makers, and Firmware Makers intending to load their Firmware into a TRSH.

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://globalplatform.org/specifications/ip-disclaimers/. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

The tables below list references applicable to this specification. The latest version of each reference applies unless a publication date or version is explicitly stated.

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPC_SPE_140	GlobalPlatform Technology Virtual Primary Platform – Network Protocol	[VNP]
GPC_SPE_141	GlobalPlatform Technology Virtual Primary Platform – OFL VNP Extension	[VOFL]
GPC_SPE_142	GlobalPlatform Technology Virtual Primary Platform – Concepts and Interfaces	[VCI]
GPC_SPE_143	GlobalPlatform Technology Virtual Primary Platform – VPP Firmware Format v2.0	[VFF]
Advanced Encryption Standard (AES)	Federal Information Processing Standards Publication 197: Specification for the Advanced Encryption Standard (AES) Available at FIPS Pub 197	[FIPS-197]
ANSI X9.63:2011	Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, 2011	[ANSI X9.63]
ANSI X9.95	Trusted Time Stamp Management and Security	[ANSI X9.95]
ANSSI ECC	Journal officiel électronique authentifié n° 0241 du 16/10/2011	[ANSSI ECC]
Brainpool	ECC Brainpool Standard Curves and Curve Generation v.1.0 19.10.2005	[Brainpool]
BSI TR-03111	BSI Technical Guideline TR-03111: Elliptic Curve Cryptography. Version 2.0	[TR 03111]
FIPS PUB 180-4:2015	https://csrc.nist.gov/publications/detail/fips/180/4/final SHA-256: Specifications for the Secure Hash Standard	[FIPS 180-4]
FIPS 186-4	FIPS PUB 186-4 Digital Signature Standard (DSS) July 2013	[FIPS 186-4]
GDPR	General Data Protection Regulation	[GDPR]
ISO/IEC 2382:2015	Information technology – Vocabulary	[2382]
RFC 1630	Universal Resource Identifiers in WWW	[RFC 1630]
RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
RFC 3647	Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework	[RFC 3647]
RFC 4122	A Universally Unique IDentifier (UUID) URN Namespace	[RFC 4122]

Standard / Specification	Description	Ref
RFC 4210	Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)	[RFC 4210]
RFC 5280	Internet X.509 PKI Certificate and CRL Profile	[RFC 5280]
RFC 5480	Elliptic Curve Cryptography Subject Public Key Information	[RFC 5480]
RFC 5639	Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation	[RFC 5639]
RFC 5758	Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA	[RFC 5758]
RFC 5759	Suite B Certificate and Certificate Revocation List (CRL) Profile	[RFC 5759]
RFC 6960	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP	[RFC 6960]
RFC 8141	Uniform Resource Names (URNs)	[RFC 8141]
SIMA	TCA – Image Delivery Server (IDS) to Open Firmware Loader (OFL) Agent Interfaces	[SIMA]

Table 1-2: Informative References

Reference	Description	Ref
BSI-CC-PP-0084-2014	Security IC Platform BSI Protection Profile 2014 with Augmentation Packages	[PP-0084]
Advances in Cryptology	Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012.* Proceedings, chapter Breaking and Repairing GCM Security Proofs, pages 31–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.	[CRYPTO 2012]
INDOCRYPT 2004	David A. McGrew and John Viega. *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004.* Proceedings, chapter The Security and Performance of the Galois/Counter Mode (GCM) of Operation, pages 343–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.	[INDO 2004]

1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY in this document (refer to [RFC 2119]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** and **SHOULD NOT** indicate recommendations.
- **MAY** indicates an option.

Selected terms used in this document are included in Table 1-3.

Table 1-3: Terminology and Definitions

Term	Definition
Actor	Legal entity involved in the OFL ecosystem either as the provider of a process in the OFL data flow or as a provider of assets (e.g. keys, certificates, etc.).
Batch Number (BN)	A non-uniform code that unambiguously identifies a particular batch of Secure Hardware Platforms sharing the same manufacturing operation and the same Part Number. By uniquely identifying a batch in a particular manufacturer organization, this code allows the tracing of Secure Hardware Platforms and the review of all stages of their production flow.
Batch Number Pseudonym Identifier (BNPI)	A universally unique and uniform pseudonym identifier of the Batch Number.
Certificate	Digital Public Key Certificate as defined in [RFC 5280]. Only two classes of Certificate exist: CA Certificates and End Entity Certificates as defined in [RFC 5280] section 3.2.
Certificate Issuer (CI)	Trust anchor of the CA.
Certificate Policy	A set of rules that indicate the applicability of a named Certificate to a particular community and/or PKI implementation with common security requirements.
Certification Authority (CA)	Actor generating Certificates.
Certification Path	List of hierarchically chained Certificates that is validated by the certification path validation algorithm defined in [RFC 5280].
CODE_M	Code that is tested to ensure that it matches a pending Image within an IDS.
Deterministic Secure Hardware Platform Pseudonym Identifier (DET_PI)	Secure Hardware Platform Pseudonym Identifier that can be regenerated by the OFL from a deterministic seed.
End Entity Certificate	Certificate issued to a subject that is not authorized to issue Certificates, as defined in [RFC 5280].
Ephemeral Secure Hardware Platform Pseudonym Identifier (EPH_PI)	Secure Hardware Platform Pseudonym Identifier that cannot be regenerated by the OFL.

Term	Definition
Firmware	In the context of this document, binary data to be loaded into the non-volatile memory managed by the TRSH; may contain an Operating System and applications or binary data belonging to the Secure Hardware Platform or the OFL.
Firmware Group	Firmware containing non-diversified data that can be installed on a Group of TRSH.
Firmware Loading	Operation performed by the Open Firmware Loader to extract a Firmware from an Image and to load this Firmware into a main storage of the TRSH.
Firmware Maker	Actor creating the Firmware to address a use case (e.g. an Operating System, its applications, and associated parameters) according to the directives of an Image Owner.
Firmware Segment	Part of a Firmware.
Firmware Session	Lifetime of the Firmware, starting with Firmware Loading and ending with Firmware Deletion. Data associated with the Firmware Session is needed for Firmware Management.
Firmware Update	Operation performed by the Open Firmware Loader to update some parts of a loaded Firmware.
Function	A set of operations performed in compliance with requirements.
Image	Generic data container encapsulating an encrypted Firmware version and its cryptographic data to be used by the OFL.
Image Binder Descriptor	Data structure providing information related to the encryption of the Image Descriptor.
Image Delivery Server (IDS)	Function in charge of binding an Unbound Image to an OFL for a given TRSH then providing the resulting bound Image to the OFL Agent.
Image Descriptor (IMD)	Data structure providing information related to the Firmware.
Image Loading	Operation performed by the OFL Agent to load an Image into the Open Firmware Loader.
Image Maker	Actor creating the Image – including the Firmware and the associated parameters – according to the directives of an Image Owner.
Image Owner (IMO)	Actor defining the requirements of a Firmware and an Image.
Image Segment	Encrypted Firmware Segment.
Image Session	Starts with the generation of the credentials required for the binding of an Image and ends with the Firmware Loading.
Image Session Data	Credentials and other data associated with an Image Session.
Image Session Identifier (ISID)	Identifier of the Image Session.
Immutable Secure Hardware Platform Pseudonym Identifier (IMU_PI)	Deterministic Secure Hardware Platform Pseudonym Identifier that can be regenerated by the OFL and requires no seed.

Term	Definition
Local OFL Operation	An OFL Operation resulting from a local trigger and not involving any communication with the IDS.
Manifest	Data containing the necessary instructions related to the Image structure in order for the OFL Agent to load the Image properly into the OFL.
OFL Agent	Software running on the device, managing the communication with the IDS and the transfer of the Image to the Open Firmware Loader.
OFL Authority	Actor granting the rights to an Image Owner allowing its Image to be accepted by OFL.
OFL Firmware	Firmware needed for the data and the execution of the OFL.
OFL Operation	An operation performed by the OFL, as listed in section 8.6.
Open Firmware Loader (OFL)	Generic/open loader allowing the provisioning of use case dependent Firmware. The provisioning can concern part of an existing Firmware (Firmware Update) or a whole Firmware (subsequent or initial Firmware Loading). The Open Firmware Loader runs on the Secure Hardware Platform and enables the activation of a Firmware from the Secure Hardware Platform.
Operation Token	Cryptographic token used as a proof of the completion of an OFL Operation.
Part Number	A non-uniform code that identifies a particular Secure Hardware Platform together with the Open Firmware Loader as assigned by the TRSH Maker.
Part Number Identifier	Universally unique and uniform identifier determined from the Part Number and the TRSH Maker identifier.
Regular Execution Environment (REE)	An Execution Environment comprising at least one Regular OS and all other components of the device (SoCs, other discrete components, firmware, and software) which execute, host, and support the Regular OS (excluding any Secure Components included in the device). From the viewpoint of a Secure Component, everything in the REE is considered untrusted, though from the Regular OS point of view there may be internal trust structures.
Root CA	CA managing a public key that originates one or more Certification Paths and acts as a trust anchor, as defined in [RFC 5280].
Secure Hardware Platform	Hardware platform and low level software within a TRSH; agnostic about the use cases and the Firmware loaded by OFL. The Secure Hardware Platform supports the OFL function.
Secure Hardware Platform Pseudonym Identifier (SHPPI)	A universally unique and uniform pseudonym identifier of the Secure Hardware Platform.
Segment Descriptor Structure (SDS)	Data containing information necessary to decrypt and manage an Image Segment.
Serial Number (SN)	A 64-bit number assigned by the TRSH Maker during the manufacturing process to a part (Secure Hardware Platform). This number is unique in the scope of the Batch Number assigned by the TRSH Maker.

Term	Definition
Tamper Resistant Secure Hardware (TRSH)	Hardware designed to isolate and protect embedded software and data by implementing appropriate security measures. The hardware and embedded software meet the requirements of the latest Security IC Platform Protection Profile (e.g. [PP-0084]) including resistance to physical tampering scenarios described in that Protection Profile.
TRSH Maker	Actor issuing the TRSH and allowed to manage/update the Secure Hardware Platform Firmware.

1.5 Abbreviations and Notations

For the purposes of this document, the following conventions apply:

- All lengths are presented in bytes, unless otherwise stated. Each byte is represented by bits b8 to b1, where b8 is the most significant bit and b1 is the least significant bit. In each representation, the leftmost bit is the most significant bit.
- Hexadecimal values are specified between single quotes, e.g. '1F'.
- All bytes specified as RFU shall be set to '00' and all bits specified as RFU shall be set to 0.

Note: Most UUIDs defined in this document are UUID version 3 or 5 as defined in [RFC 4122].

Selected abbreviations used in this document are included in Table 1-4.

Notations used in this document are defined in Table 1-5.

Table 1-4: Abbreviations

Abbreviation	Meaning
ACME	Automatic Certificate Management Environment
AKID	Authority Key IDentifier
ARP	Access Right Pattern
ATK	Authentication Token
BN	Batch Number
BNPI	Batch Number Pseudonym Identifier
CA	Certification Authority
CI	Certificate Issuer
CODE_M	See Table 1-3.
DET_PI	Deterministic Secure Hardware Platform Pseudonym Identifier
ECC	Elliptic Curve Cryptography
ECC-256	Elliptic Curve Cryptography 256-bit defined in [TR 03111]
ECC-384	Elliptic Curve Cryptography 384-bit defined in [FIPS 186-4] and [Brainpool]
ECDHE	Elliptic Curve Diffie–Hellman Ephemeral

Abbreviation	Meaning
ECDSA	Elliptic Curve Digital Signature Algorithm
ECKA	Elliptic Curve Key Agreement algorithm
ECKA-DH	Diffie-Hellman Elliptic Curve Key Agreement defined in [TR 03111]
ECKA-EG	El-Gamal Elliptic Curve Key Agreement defined in [TR 03111]
EE	End Entity (Certificate) as defined in [RFC 5280]
EPH_PI	Ephemeral Secure Hardware Platform Pseudonym Identifier
FAC	FACtory
GDPR	General Data Protection Regulation defined in [GDPR]
HSM	Hardware Security Module
IDS	Image Delivery Server
IMD	Image Descriptor
IMO	Image Owner
IMU_PI	Immutable Secure Hardware Platform Pseudonym Identifier
ISID	Image Session IDentifier
KDF	Key Derivation Function
NID	Namespace IDentifier
NSS	Namespace Specific String
OCSP	Online Certificate Status Protocol
OFL	Open Firmware Loader
OID	Object Identifier
OS	Operating System
PKI	Public Key Infrastructure
PI	Pseudonym Identifier
PN	Chip Part Number
PN CA	Part Number CA, independent of the equipment signing the CERT.OFL.ECDSA certificates
REE	Regular Execution Environment
RFU	Reserved for Future Use
SDO	Standards Development Organization
SDS	Segment Descriptor Structure
SHA-256	Hash Function defined in [FIPS 180-4]
SHPPI	Secure Hardware Platform Pseudonym Identifier
SKID	Subject Key Identifier
SN	Serial Number

Abbreviation	Meaning
SSD	Security Scheme Descriptor
TRSH	Tamper Resistant Secure Hardware
TSA	Time Stamp Authority
URN	Uniform Resource Name
UUID	Universally Unique IDentifier
VPP	Virtual Primary Platform

Table 1-5: Notations

Notation	Meaning
$[XX]_N$	N times the value ‘XX’.
{eSK,ePK}= ECDHE ()	Generate an ephemeral ECDHE key pair.
ATK.X.ECKA	Authentication Token for key agreement (dynamic Certificate of X for key agreement).
IDS_{ID}	ID denotes 1 or 2: Identifier of the sub part of the IDS (IDS_1 and IDS_2).
M=CIPHER(Y)[X]	Cryptographic primitive (e.g. AES) to encrypt X with the key Y. The result is the cryptogram M.
X=ICIPHER(Y)[M]	Cryptographic primitive to decrypt M with the key Y. The result is the plaintext X.
CERT.X.ECDSA	ECDSA static Certificate of X.
CERT.X.ECKA	ECKA static Certificate of X for key agreement.
DECRYPT(Y) [X]	Decrypt ciphertext X with the key Y.
DERIVE(X)[Y]	Compute a shared session key from a private key X and a Certificate/authentication token Y.
ENCRYPT(Y) [X]	Encrypt plaintext X with the key Y.
ePK.X.ECKA	Ephemeral ECKA public key of X.
PK.X.ECDSA	Static ECDSA public key of X for signing signature.
PK.X.ECKA	Static ECKA public key of X for key agreement.
SIGN(Y) [X]	Sign X with the key Y.
eSK.X.ECKA	Ephemeral ECKA private key of X.
SK.X.ECDSA	Static ECDSA private key of X for signing signature.
SK.X.ECKA	Static ECKA private key of X for key agreement.
VERIFY(Y) [X]	Verify the signature X with the key Y.

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-6: Revision History

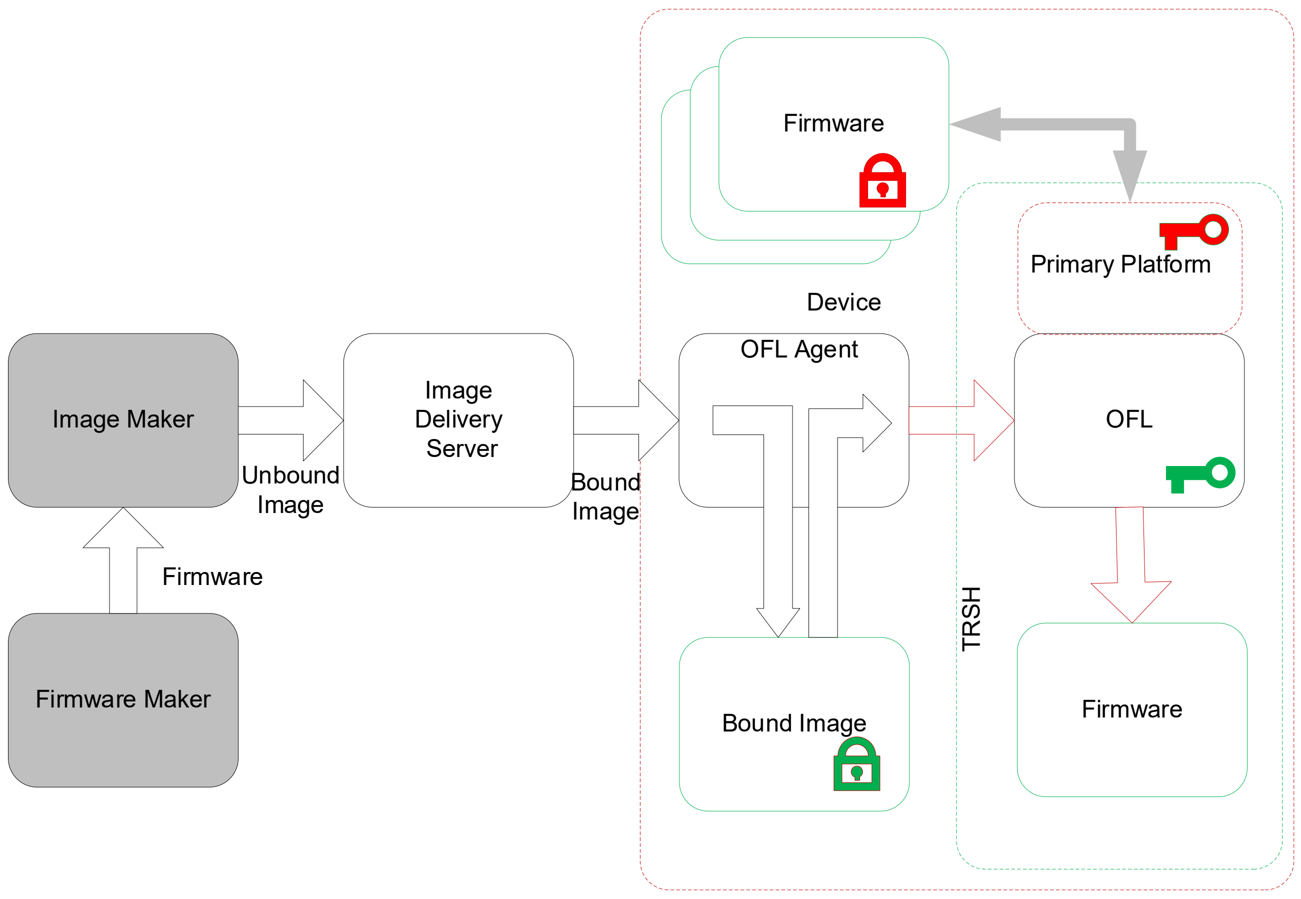
Date	Version	Description
June 2017	1.3	Public Release as *Open Firmware Loader for Tamper Resistant Element*
Nov 2019	1.3.0.8	Committee Review
Jan 2020	1.3.0.9	Member Review
Oct 2020	1.3.0.10	Member Review
April 2021	1.3.0.14	Public Review
TBD	2.0	Public Release as *Open Firmware Loader for Tamper Resistant Secure Hardware* Editorial changes: o The layout of the specification better expresses the sequence of Image Loading and Firmware management (e.g. Loading, Update, Enable, Disable, Delete…). o The notation related to the cryptographic credentials has been aligned with GlobalPlatform conventions. o Some variables have been renamed for easier understanding. o General clarifications. Technical changes: o The GlobalPlatform Certificate format has been replaced by X.509 Certificate format version 3. o Asymmetric PKI is supported. o The capability to attach a new IDS Certificate Issuer after issuance of the TRSH is supported. o Definition of Tokens is now based on the X.509 Certificate format. o Generic extensions in the Token similar to the X.509 Certificate to convey signed data which may either be in plaintext or encrypted. o ARP filter extensions have been added. o OFL Operation and related section and links have been added. o Support for new cryptography algorithms (Chinese cryptography based on SM series) has been added. o The ARP Token has been replaced by an ARP Certificate, which is easier to issue. o A Certificate revocation mechanism has been added. The Image Body format as defined in section 7.4.1 is backward compliant with v1.3 of this specification.

2 Overview

2.1 OFL Flow

Figure 2-1 presents the general overview of flow of Open Firmware Loader (OFL) data and stakeholders that will be involved when loading an Image to the OFL as defined in section 3, then managing the Firmware in the Tamper Resistant Secure Hardware (TRSH) as defined in section 8.

Figure 2-1: OFL Flow Overview



The transport protocol supporting the OFL data flow between the Image Delivery Server (IDS) and the OFL Agent is out of scope of this document.

The transport protocol supporting the OFL data flow between the OFL Agent and the OFL is out of scope of this document (e.g. *GlobalPlatform Virtual Primary Platform – OFL VNP Extension* – [VOFL]).

The transport protocol supporting the OFL data flow between the Image Maker and the Image Delivery Server is out of scope of this document.

2.2 OFL Data

This section describes the non-volatile storages containing OFL data for the OFL Operations.

The OFL manages three private and confidential non-volatile storages for OFL data:

- A Firmware Session storage containing a collection of Firmware Session data defined in section 8.5.
- An Image Session storage containing a collection of Image Session Data defined in section 3.2.
- An OFL management storage containing data for managing the OFL state, the current local Access Right Pattern (ARP), and the ARP state defined in section 8.4.

2.3 Identifiers

2.3.1.1 Deterministic Identifiers

2.3.1.1.1 URN-based Deterministic Identifiers

These identifiers are universally unique, deterministic, and uniform. Each shall be a UUID as defined in [RFC 4122] using the domain name system namespace and shall be generated from names.

These identifiers shall be version 3 or 5 UUIDs derived from the URN using the syntax defined in [RFC 8141]:

- Open Firmware Loader Type Identifier (OFL_TYPE_UUID):
  - NID is [Namespace Identifier of the TRSH Maker] and
    NSS is the type reference assigned by the TRSH Maker.
- Firmware Group Identifier (UUIDG):
  - NID is [Namespace Identifier of the Firmware Maker] and
    NSS is the Firmware Group assigned by the Firmware Maker.
- Firmware Identifiers ($UUID_P$ and $UUID_I$):
  - NID is [Namespace Identifier of the Firmware Maker] and
    NSS is the Firmware reference assigned by the Firmware Maker.
- Firmware Maker Identifier ($UUID_M$):
  - NID is [Namespace Identifier of the Firmware Maker] and NSS is the name of the Firmware Maker.
- The Part Number Identifier (PART_NUMBER):
  - NID is [Namespace Identifier of the TRSH Maker] and
    NSS is the Part Number assigned by the TRSH Maker.
- Firmware Family Identifier ($UUID_F$):
  - NID is [domain.name] or NID is [SDO domain.name] and
    NSS is the Firmware Family name assigned by the SDO.

In the context of this document, the following definitions apply:

- [domain.name]: The Fully Qualified Domain Name (FQDN) of the organization defining the URN (e.g. BananaTel.com).
- [SDO domain.name]: The FQDN of a standards development organization or an industry organization (e.g. globalplatform.org).

Note: The UUID computation is case sensitive with respect to the URN provided, including the domain name.

2.3.1.1.2 Other Deterministic Identifiers

These identifiers are universally unique, deterministic, and uniform. They are 128-bit identifiers and do not support the format defined in [RFC 4122].

- Image Owner (IMO) Identifier ($UUID_{IMO}$):
  - By using the content of CERT.IMO.ECDSA, this identifier is composed of the 128 most significant bits of the Subject Key Identifier (SKID) from the SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits) as defined in [RFC 5280] section 4.1.1.2.
- OFL Authority Identifier ($UUID_{OFLA}$):
  - This identifier is composed of the 128 most significant bits of the SKID from the SHA-1 hash of the value of the BIT STRING $PK.IMO_{OFLA}$.ECDSA (excluding the tag, length, and number of unused bits).
- TRSH Maker Authority Identifier:
  - This identifier is composed of the 128 most significant bits of the SKID from the SHA-1 hash of the value of the BIT STRING $PK.TRSH_{M}$.ECDSA (excluding the tag, length, and number of unused bits).
- Deterministic Secure Hardware Platform Pseudonym Identifier (DET_PI): defined in section 10.2.
- Immutable Secure Hardware Platform Pseudonym Identifier (IMU_PI): defined in section 10.2.

2.3.1.2 Ephemeral Identifier

These identifiers are universally unique, ephemeral, and uniform. Each shall be a UUID as defined in [RFC 4122] and may be generated from names.

These identifiers shall be version 4 or 5 UUIDs derived from the URN using the syntax defined in [RFC 8141] or from random:

- Image Session Identifier (ISID):
  - NID is [Namespace Identifier of the TRSH Maker] and NSS is value assigned and only known by the OFL. This value may be determined from a monotonic counter (UUID version 5) or from a random number (UUID version 4). This identifier is generated by OFL.

2.3.1.3 Predefined Identifiers

Table 2-1 describes the pre-defined 128-bit identifiers for the OFL Authority. Those values are not computed as described in previous sections.

Table 2-1: Pre-defined OFL Authority Identifier

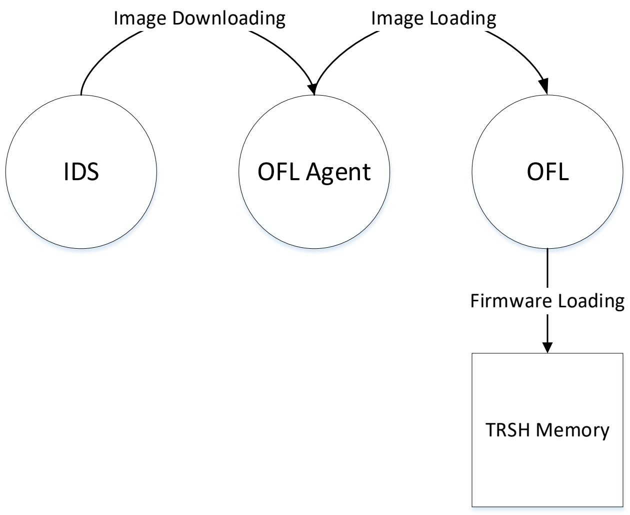
Name	Value
$OFLA\_ID_{BLANK}$	FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF
$OFLA\_ID_{UNLOCKED}$	00000000-0000-0000-0000-000000000000

3 Image Journey

3.1 Overview

This section describes the Image journey and the consecutive steps that occur until the Firmware is loaded in the TRSH Memory. Subsequent to the downloading of the Image from the IDS to the OFL Agent, the Image Loading is the operation before the Firmware Loading, as shown in Figure 3-1.

Figure 3-1: Image Journey



The Image Loading is the transfer of the Image from the OFL Agent to the OFL. The Firmware Loading is the transfer of the Firmware from the OFL to the TRSH Memory. The process of Firmware Loading and the process of Image Loading are performed at the same time and are protected by a transaction mechanism. Therefore, the Firmware Loading is completed when the Image Loading is completed.

The protocol between the OFL and OFL Agent is out of scope of this document. One example for this protocol is the *GlobalPlatform VPP Network Protocol Extension* ([VOFL]).

It shall support the following mandatory commands sent by the OFL Agent to the OFL:

- **ANY_SET_PARAMETER:** Set data to the OFL registry.
- **ANY_GET_PARAMETER:** Get data from the OFL registry.
- **OFL_DO_OPERATE:** Transfer the OFL_DO_OPERATE parameter used in all the operations referenced in section 8.6.
- **OFL_CHANGE_SEGMENT:** Transfer the OFL_CHANGE_SEGMENT parameter.
- **OFL_LOAD_SEGMENT:** Transfer an OFL_LOAD_SEGMENT parameter containing an Image Segment.

It shall support the following mandatory responses sent from the OFL to the OFL Agent:

- **ANY_OK:** Successful execution.
- **ANY_NOK:** Failed execution.

Each of the above commands supports a parameter as defined in sections 6.2.2, 6.2.3, and 6.3.

The protocol between the IDS and the OFL Agent is out of scope of this document. One example is described in *TCA – Image Delivery Server (IDS) to Open Firmware Loader (OFL) Agent Interfaces* ([SIMA]).

3.2 Image Session Data

Several Image Sessions and their associated data can coexist simultaneously (the maximum number of sessions is implementation dependent).

The Image Session Data dedicated to a pending Image as described in section 5.4.2.

An Image Session starts with the execution of ANY_GET_PARAMETER to get the TRSH Credentials (see Figure 5-1) from the OFL.

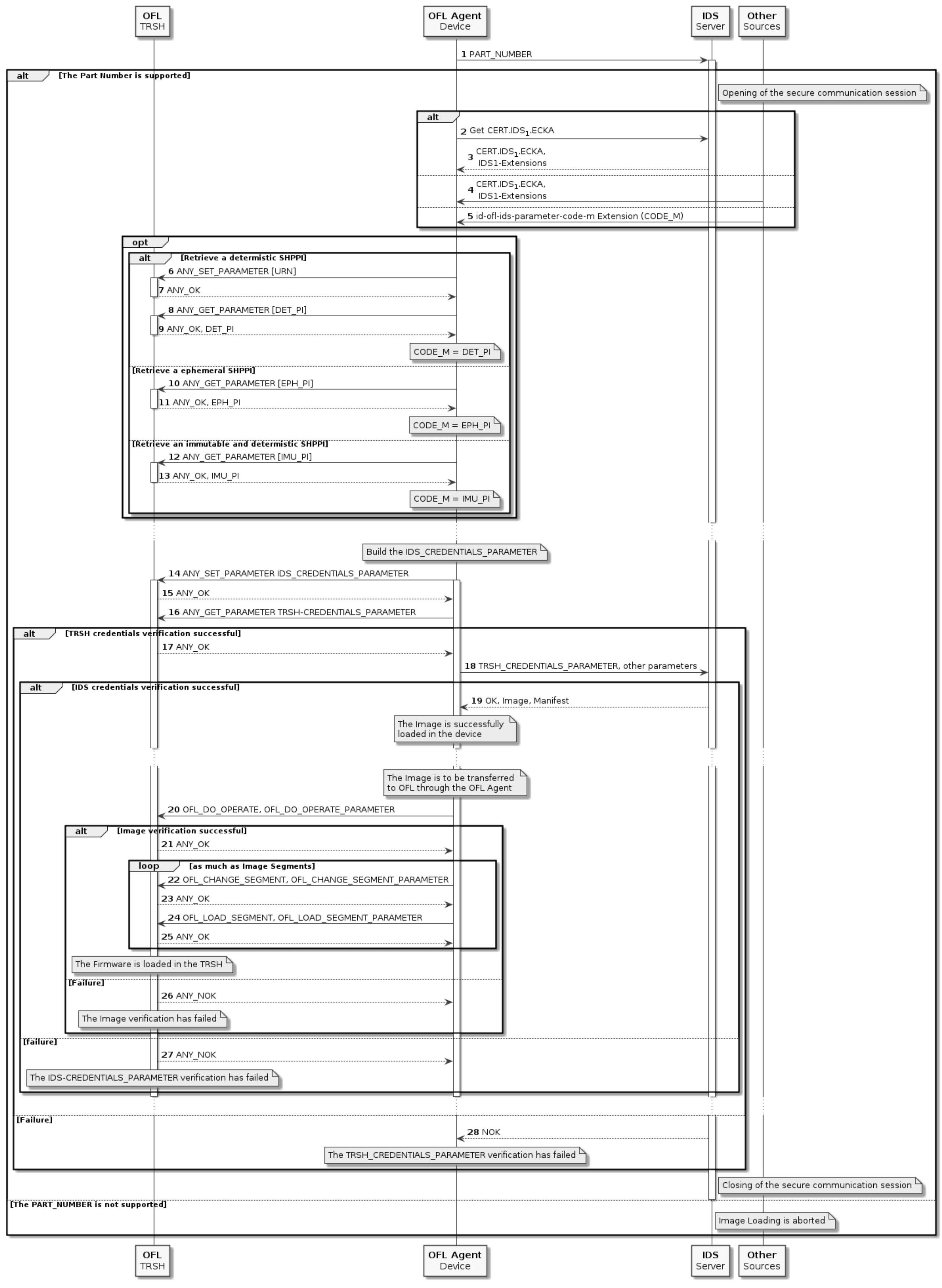
An Image Session is stopped then discarded:

- Once the corresponding Image has been successfully loaded; or
- when this Image Session is overwritten by a new Image Session when the number of concurrent Image Sessions exceeds a given limit.

3.3 Image Loading Procedure

Figure 3-2 shows an overview of the functional flow between the OFL, the OFL Agent running in the Regular Execution Environment (REE) of the device, and the IDS which resides outside the device. Details of the security protocol are explained in section 5 of this document.

Figure 3-2: Image Loading Procedure

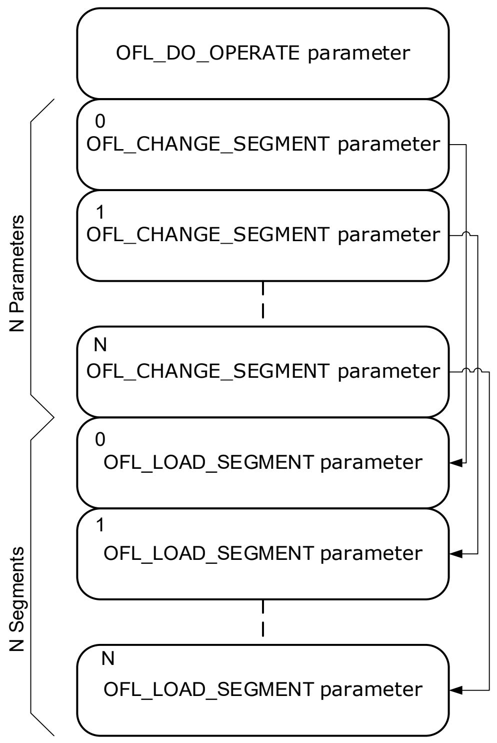

OFL TRSH
OFL Agent Device
IDS Server
Other Sources
1 PART_NUMBER
alt [The Part Number is supported]
Opening of the secure communication session
alt
2 Get CERT.IDS1.ECKA
3 CERT.IDS1.ECKA, IDS1-Extensions
4 CERT.IDS1.ECKA, IDS1-Extensions
5 id-ofl-ids-parameter-code-m Extension (CODE_M)
opt
alt [Retrieve a determistic SHPPI]
6 ANY_SET_PARAMETER [URN]
7 ANY_OK
8 ANY_GET_PARAMETER [DET_PI]
9 ANY_OK, DET_PI
CODE_M = DET_PI
[Retrieve a ephemeral SHPPI]
10 ANY_GET_PARAMETER [EPH_PI]
11 ANY_OK, EPH_PI
CODE_M = EPH_PI
[Retrieve an immutable and determistic SHPPI]
12 ANY_GET_PARAMETER [IMU_PI]
13 ANY_OK, IMU_PI
CODE_M = IMU_PI
Build the IDS_CREDENTIALS_PARAMETER
14 ANY_SET_PARAMETER IDS_CREDENTIALS_PARAMETER
15 ANY_OK
16 ANY_GET_PARAMETER TRSH-CREDENTIALS_PARAMETER
alt [TRSH credentials verification successful]
17 ANY_OK
18 TRSH_CREDENTIALS_PARAMETER, other parameters
alt [IDS credentials verification successful]
19 OK, Image, Manifest
The Image is successfully loaded in the device
The Image is to be transferred to OFL through the OFL Agent
20 OFL_DO_OPERATE, OFL_DO_OPERATE_PARAMETER
alt [Image verification successful]
21 ANY_OK
loop [as much as Image Segments]
22 OFL_CHANGE_SEGMENT, OFL_CHANGE_SEGMENT_PARAMETER
23 ANY_OK
24 OFL_LOAD_SEGMENT, OFL_LOAD_SEGMENT_PARAMETER
25 ANY_OK
The Firmware is loaded in the TRSH
[Failure]
26 ANY_NOK
The Image verification has failed
[failure]
27 ANY_NOK
The IDS-CREDENTIALS_PARAMETER verification has failed
[Failure]
28 NOK
The TRSH_CREDENTIALS_PARAMETER verification has failed
Closing of the secure communication session
[The PART_NUMBER is not supported]
Image Loading is aborted
OFL TRSH
OFL Agent Device
IDS Server
Other Sources

The Image Loading procedure relies on the structure of the Image handled by the OFL Agent. The structure of the Image is defined in Figure 3-3.

The procedure to download the Image and its associated Manifest from the IDS to the OFL Agent is out of scope of this document.

The unicast Image Loading procedure shall start at step 1. The multicast Image Loading procedure shall start at step 14.

Figure 3-3: Structure of the Image as Seen by the OFL Agent


OFL_DO_OPERATE parameter
N Parameters
0
OFL_CHANGE_SEGMENT parameter
1
OFL_CHANGE_SEGMENT parameter
N
OFL_CHANGE_SEGMENT parameter
N Segments
0
OFL_LOAD_SEGMENT parameter
1
OFL_LOAD_SEGMENT parameter
N
OFL_LOAD_SEGMENT parameter

The content of the Image parameters, as defined in Figure 3-3, is transparent to the OFL Agent, which retrieves these parameters and uses the associated Manifest to combine them with their respective command, then applies the procedure as defined in Figure 3-2.

The Image Loading procedure is agnostic about the security scheme or the content of the Image Segment(s).

The OFL Agent does not perform any cryptographic operations and manages neither confidential keys nor credentials / secret data related to the Image Loading.

The procedure defined in Figure 3-2 is the following:

1. The OFL Agent initiates a secure communication session with the IDS for a given PART_NUMBER. If the PART_NUMBER is not supported by the IDS, then the Image Loading aborts.
2. The OFL Agent requests from the IDS the CERT.IDS$_1$.ECKA and the optional IDS1-Extensions (e.g. see [SIMA]).
3. The IDS returns the CERT.IDS$_1$.ECKA and the optional IDS1-Extensions and additional parameters if required (e.g. see [SIMA]).

4. As an alternative to steps 2 and 3, the OFL Agent receives, by any means other than a request to the IDS (e.g. NFC NDEF, QR-CODE, Push, etc.), the CERT.IDS$_1$.ECKA and the optional IDS1-Extensions and additional parameters if required. The way this information is provided is out of scope of this document.
5. As a last alternative, the OFL Agent receives CODE_M in the optional IDS1-Extensions. The way this information is provided is out of scope of this document.
6. The OFL Agent provides a URN to the OFL in order to request the computation of a Secure Hardware Platform Pseudonym Identifier (SHPPI) as defined in section 10.2.
7. The OFL returns ANY_OK.
8. The OFL Agent optionally requests a DET_PI (deterministic SHPPI) as defined in section 10.2 from the OFL registry.
9. The OFL returns the DET_PI value.
10. As an alternative to steps 6-9, the OFL Agent optionally requests an EPH_PI (Ephemeral SHPPI) as defined in section 10.2 from the OFL registry.
11. The OFL returns the EPH_PI value.
12. As a last alternative, the OFL Agent optionally requests an IMU_PI (immutable and deterministic SHPPI) as defined in section 10.2 from the OFL registry.
13. The OFL returns the IMU_PI value.
14. The OFL Agent builds the IDS_CREDENTIALS_PARAMETER parameter as defined in section 6.2.2. The OFL Agent sends the IDS_CREDENTIALS_PARAMETER parameter to the OFL registry entry IDS_CREDENTIALS_PARAMETER defined in section 10.3.
15. The OFL returns ANY_OK for acknowledging the update of the IDS_CREDENTIALS_PARAMETER parameter.
16. The OFL Agent requests the TRSH_CREDENTIALS_PARAMETER data as defined in section 10.3, from the TRSH_CREDENTIALS_PARAMETER registry entry content as defined in section 10.3.
17. The OFL verifies the IDS_CREDENTIALS_PARAMETER parameter, generates the TRSH_CREDENTIALS_PARAMETER data, then returns ANY_OK if the verification is successful.
18. The OFL Agent sends the TRSH_CREDENTIALS_PARAMETER data to the IDS (e.g. see [SIMA]).
19. The IDS returns OK, as well as the Image and the Manifest containing the instructions for the Image Loading from the OFL Agent to the OFL (e.g. see [SIMA]).
20. The OFL Agent extracts the OFL_DO_OPERATE_PARAMETER from the Image. The OFL Agent sends to the OFL the OFL_DO_OPERATE command with its parameter (OFL_DO_OPERATE_PARAMETER).
21. The OFL performs the processing of the OFL_DO_OPERATE command. If the processing is successful, then the OFL returns ANY_OK to the OFL Agent.
22. The OFL Agent extracts the OFL_CHANGE_SEGMENT_PARAMETER parameter from the Image and sends the OFL_CHANGE_SEGMENT command using its parameter (OFL_CHANGE_SEGMENT_PARAMETER).
23. The OFL performs the processing of the OFL_CHANGE_SEGMENT command. If the processing is successful, then the OFL returns ANY_OK to the OFL Agent.
24. The OFL Agent extracts the OFL_LOAD_SEGMENT_PARAMETER parameter from the Image and sends the OFL_LOAD_SEGMENT command using its parameter (OFL_LOAD_SEGMENT parameter).

25. The OFL performs the processing of the OFL_LOAD_SEGMENT command. If the processing is successful, then the OFL returns ANY_OK to the OFL Agent. At the reception of this status for the last Image Segment, the Image and the Firmware are loaded.
26. If the OFL fails at processing the OFL_DO_OPERATE command. The OFL returns ANY_NOK to the OFL Agent.
27. If the IDS fails at processing the TRSH_CREDENTIALS_PARAMETER data. The IDS returns NOK to the OFL Agent.
28. The OFL fails at processing the IDS_CREDENTIALS_PARAMETER. The OFL returns ANY_NOK to the OFL Agent.

Steps 6 to 13 apply only if the IDS1-Extensions, which the OFL Agent received in the preceding steps, included CODE_M.

Steps 22 to 25 shall be repeated until all segments of the Image are successfully loaded. The number of segments of the Image is defined in its Manifest.

An Image may have no Image Segment (and therefore no OFL_CHANGE_SEGMENT_PARAMETER and OFL_LOAD_SEGMENT_PARAMETER). In such a case, only the OFL_DO_OPERATE command is sent to the OFL; this instructs the OFL to perform administrative operations. Consequently, the Image Loading is completed at step 21.

Steps 14 and 17 are paired. The TRSH_CREDENTIALS_PARAMETER data are generated by the OFL for a given IDS_CREDENTIALS_PARAMETER.

The Image Loading starts at step 20 and stops at step 25.

If an OFL_DO_OPERATE command is sent during an Image Loading, then this procedure restarts at step 20.

Note: The description above does not include the notification steps; e.g. how the IDS retrieves from the OFL the OFL Operation Tokens described in section 5.5.5.

4 Security Operations and Functions

4.1 Security Operations

The following security operations can be performed:

Signing

CERT.X.ECDSA [Signature] = SIGN (SK.Y.ECDSA) [PK.X.ECDSA, X0, X1, X2, X3,…, XN]

CERT.X.ECKA [Signature] = SIGN (SK.Y.ECDSA) [PK.X.ECKA, X0, X1, X2, X3…, XN]

ATK.X.ECKA = SIGN (SK.Y.ECDSA) [ePK.X.ECKA, X0, X1, X2, X3,…, XN]

where:

X0, X1, X2, X3,.., XN are optional values to sign.

The SIGN(Y) [X] function is described in section 4.2.4.

Verification

VERIFY (PK.Y.ECDSA) [CERT.X.ECDSA]

VERIFY (PK.Y.ECDSA) [ATK.X.ECKA]

returns a Boolean (true when successful) by using PK.X.ECDSA.

The VERIFY(Y)[X] function is described in section 4.2.5.

Derivation

$KS\{Y\}^N$ = $DERIVE^N$ (A.Z.ECKA) [B.X.ECKA]

Where:

A	:	eSK or SK: an ephemeral or a static secret key
B	:	ePK or PK: an ephemeral public key or a static public key
Z and X	:	The name of a Certificate or an Authentication Token
$KS\{Y\}^N$	:	A vector of keys with three components (K, IV, EIV) and referred to as a key matrix. Its structure is detailed in section 4.2.2 (i.e. $KS1^9$, $KS2^9$, $KS3^9$).
N	:	The number of elements in the key matrix KS{Y}
Y	:	The reference of the keys (A and B); takes values 1, 2, 3

The $DERIVE^N$ (Z) [X] function is described in section 4.2.1.

Encryption / Decryption

M_{DEST}, H_{SRC} = ENCRYPT (MKS) [M_{SRC}]

Where:

MKS	:	A matrix of keys
M_{SRC}	:	The plaintext
M_{DEST}	:	The ciphertext
H_{SRC}	:	The integrity check on the plaintext

The ENCRYPT(Y) [X] function is described in section 4.2.2.

M_{DEST}, H_{SRC} = DECRYPT (MKS) [M_{SRC}]

Where:

MKS	:	A matrix of keys
M_{SRC}	:	The ciphertext
M_{DEST}	:	The plaintext
H_{SRC}	:	The integrity check on the plaintext

The DECRYPT(Y) [X] function is described in section 4.2.2.

M_{DEST} = CIPHER (KS) [M_{SRC}]

Where:

KS	:	A key
M_{SRC}	:	The plaintext block
M_{DEST}	:	The ciphertext block

The CIPHER(Y) [X] function is described in section 4.2.2.

M_{DEST} = ICIPHER(M_{SRC}) [KS]

Where:

M_{SRC}	:	The ciphertext block
M_{DEST}	:	The plaintext block
KS	:	A key

The ICIPHER(Y) [X] function is described in section 4.2.2.

4.2 Security Functions

4.2.1 Key Derivation Function

The function DERIVE^{N}(X)[Y] allows the computation of a shared secret key from the private key X and a public key Y within a Certificate or an authentication token as defined in [TR 03111]. The procedure starts from the computation of ShS as a shared secret ECKA-EG or ECKA-DH, as defined in [TR 03111], and a Key Derivation Function KDF.

KDF is an X9.63 Key Derivation Function as defined in [TR 03111] that uses a hash function (Hash(x)) and generates a derivation data as a byte array to be mapped on a structure table to obtain KS{Y}^{N} as KS{Y} a matrix of keys having a dimension of N keys.

Hash(x) shall be a SHA-256 defined in [FIPS 180-4].

KS{Y}^{N} = KDF (ShS, I, SI)

Where:

ShS	:	The 256-bit shared secret from the Key Agreement Algorithm ECKA-EG (using a static key and an ephemeral key) or ECKA-DH (using two ephemeral keys) as defined in [TR 03111].
I	:	The number of loops
SI	:	The Shared Info as SI= KEY_TYPE ^{8BIT}|| ALGORITHM ^{8BIT}|| KEY_LENGTH ^{8BIT}

ALGORITHM	:	'89' for the eGCM algorithm
KEY_LENGTH	:	The length of the key in bytes

Y	:	The reference of the key that indicates the SI and ShS used to generate the key matrix; takes values 1, 2, 3
N	:	The number of keys in the matrix (9). The key sizes depend on the table structure, as specified in section 4.2.2.

Table 4-1 defines the parameters for KDF according to the keys and their length.

Table 4-1: Key Parameters for eGCM-AES

Size of the Key	N	Y	KEY_LENGTH	KEY_TYPE	I
128 bits	9	1	'10'	'10'	5
		2	'10'	'20'	5
		3			
256 bits (Only for AES)		2	'20'	'20'	6
		3			

Table 4-2 defines the private and public keys used for generating the Shared Secrets according the reference Y of the keys KSY^N.

Table 4-2: Shared Secret ShS determination

Key Reference (Y)	Shared Secret
1	ShS from: • $PK.IDS_1.ECKA$ and eSK.OFL.ECKA • $SK.IDS_1.ECKA$ and ePK.OFL.ECKA
2	ShS from: • $ePK.IDS_2.ECKA$ and eSK.OFL.ECKA • $eSK.IDS_2.ECKA$ and ePK.OFL.ECKA
3	ShSg shared secret for Firmware Group

A subset of keys in the matrix KSY^N can be referenced with the notation KSY_a^M, where:

Y	:	The reference type of the key
a	:	The index (starting from 0) of the key subset of three keys in KSY^N
M	:	The number of keys of this subset

4.2.2 Encryption / Decryption Function

The functions ENCRYPT (X)[Y] and DECRYPT (X)[Y] are symmetrical functions based on the eGCM-AES-128/eGCM-AES-256 algorithms defined in Annex B.

$$M_{DST}, H_{SRC} = \text{eGCM-AES-128}(K, IV, EIV)[M_{SRC}]$$

$$M_{DST}, H_{SRC} = \text{eGCM-AES-256}(K, IV, EIV)[M_{SRC}]$$

Where:

K	:	The Encrypting/Decrypting and Integrity Key; a 128-bit key for eGCM-AES-128 or a 256-bit key for eGCM-AES-256
IV	:	The 128-bit Initialization Vector used for integrity check
M_{SRC}	:	The plaintext/ciphertext to encrypt/decrypt. The length of the message shall be a multiple of 128-bit blocks padded with 'FF'.
EIV	:	The 128-bit Initialization Vector used for encryption
M_{DST}	:	The encrypted/decrypted plaintext/ciphertext. The length of the ciphertext is the same as the M_{SRC} plaintext.
H_{SRC}	:	The 128-bit integrity check result of the plaintext (source)

All keys and Initialization Vectors are derived from the key derivation function (see section 4.2.1), mapping the key derivation data on the table structure related to the keys and the encryption algorithm as follows:

Table 4-3: KS1 eGCM-AES-128 Parameters

Reference	Key	eGCM-AES-128 Parameter	Size of the Field (Byte)
KS1[9]	$KS1_0$[3]	K	16
		IV	16
		EIV	16
	$KS1_1$[3]	KE	16
		IV	16
		EIV	16
	$KS1_2$[3]	K	16
		IV	16
		EIV	16
Key derivation data size			144

Table 4-4: KS2 eGCM-AES-128 Parameters

Reference	Key	eGCM-AES-128 Parameter	Size of the Field (Byte)
$KS2$[9]	$KS2_0$[3]	K	16
		IV	16
		EIV	16
	$KS2_1$[3]	KE	16
		IV	16
		EIV	16
	$KS2_2$[3]	K	16
		IV	16
		EIV	16
Key derivation data size			144

Table 4-5: KS3 eGCM-AES-128 Parameters

Reference	Key	eGCM-AES-128 Parameter	Size of the Field (Byte)
$KS3$[9]	$KS3_0$[3]	K	16
		IV	16
		EIV	16
	$KS3_1$[3]	KE	16
		IV	16
		EIV	16
	$KS3_2$[3]	K	16
		IV	16
		EIV	16
Key derivation data size			144

Table 4-6: KS2 eGCM-AES-256 Parameters

Reference	Key	eGCM-AES-256 Parameter	Size of the Field (Byte)
KS2[9]	$KS2_0$[3]	K	32
		IV	16
		EIV	16
	$KS2_1$[3]	KE	32
		IV	16
		EIV	16
	$KS2_2$[3]	K	32
		IV	16
		EIV	16
Key derivation data size			192

Table 4-7: KS3 eGCM-AES-256 Parameters

Reference	Key	eGCM-AES-256 Parameter	Size of the Field (Byte)
KS3[9]	$KS3_0$[3]	K	32
		IV	16
		EIV	16
	$KS3_1$[3]	KE	32
		IV	16
		EIV	16
	$KS3_2$[3]	K	32
		IV	16
		EIV	16
Key derivation data size			192

During operations, K, IV, and EIV are rotated every 16 blocks (16x128 bits), as described in section 4.2.3.

The function CIPHER(X) [Y] and ICIPHER(X) [Y] are block cipher functions for respectively encrypting and decrypting a block of data.

CIPHER may be AES-128 or AES-256 as defined as the cipher function in [FIPS-197].

ICIPHER may be AES-128^{-1} or AES-256^{-1} as defined as the inverse cipher function in [FIPS-197].

4.2.3 Key Rotation Function

The following operation allows rotating a key Y_N for eGCM-AES-128:

> **C_1 = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,0xFF}**
>
> **X_1 = AES-128 (Y_N) [C_1]**
>
> **Y_{N+1} = Y_N XOR X_1**

The following operation allows rotating a key Y_N for eGCM-AES-256:

> **C_2 = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }**
>
> **X_2 = AES-256 (Y_N) [C_2]**
>
> **Y_{N+1} = Y_N XOR X_2**

4.2.4 Signing Function

The function SIGN(Y) [X] signs X with the key Y corresponds to the Elliptic Curve Digital Signature Algorithm (ECDSA) Signature Algorithm defined in [TR 03111].

4.2.5 Verifying Function

The function VERIFY(Y) [X] verifies the signature X with the public key Y and corresponds to the ECDSA Verification Algorithm defined in [TR 03111].

The function VERIFY_PATH(Y) [X] validates the Certification Path originating from the public key Y and ending with the End Entity Certificate X, according to the rules defined in section 11.1.3.

5 Security Protocol

The security protocol is based on ECC-256/ECC-384 using the ECC curve domain parameters of the public key of the CERT.IDS$_1$.ECKA Certificate from the IDS.

5.1 Static Keys

5.1.1 Long-term Static Keys

The long-term static keys of the OFL and the IDS are in Table 5-1.

Table 5-1: Long-term Static Keys

Category	Key	Definition
Related to OFL	**PK.PN.ECDSA**	Public key for signature verification in the Part Number Certificate as the CA of all Factory (FAC) Certificates as defined in section 11.2.9.
	SK.PN.ECDSA	Private key for signature generation of the Part Number Certificates
	PK.FAC.ECDSA	Public key for signature verification in the FAC Certificate as defined in section 7.
	SK.FAC.ECDSA	Private key for signature generation of the OFL Certificate
	PK.OFL.ECDSA	Public key for signature verification in the OFL Certificate as defined in section 11.2.8.
	SK.OFL.ECDSA	Private key for signature generation of the ATK.OFL.ECKA Authentication Tokens.
	PK.CI$_{OFL}$.ECDSA	Public key for signature verification in the CI Certificate as defined in section 11.2.1. This public key is the trust anchor of the OFL Certification Path.
	PK.CI$_{IDS}$.ECDSA	Public key is the trust anchor of the IDS Certification Path.
	PK.TRSH$_M$.ECDSA	Public key for signature verification of the ARP Certificate from the TRSH Maker for loading/updating Secure Hardware Platform Firmware
	PK.OFLA.ECDSA	Public key for signature verification of the ARP Certificate from the OFL Authority or loading/updating Secure Hardware Platform Firmware
Related to IDS	**PK.IDS.ECDSA**	Public key for signature verification in the IDS$_1$ and IDS$_2$ Certificates as defined in sections 11.2.5, 11.2.6, and 11.2.7.
	SK.IDS.ECDSA	Private key for signature generation of the IDS$_1$ and IDS$_2$ Certificates.
	PK.IDS$_2$.ECDSA	Public key for signature verification in the IDS$_2$ Certificate as defined in section 11.2.7.

Category	Key	Definition
	SK.IDS$_2$.ECDSA	Private key for signature generation of the ATK.IDS$_2$.ECKA Authentication Tokens. This private key of the IDS for signature is hosted in the HSM_IDS$_2$ (Hardware Security Module 2 of IDS).
	PK.IDS$_1$.ECKA	Public key for key agreement in the IDS$_1$ Certificate as defined in section 11.2.5.
	SK.IDS$_1$.ECKA	Private static key of the IDS for key agreement hosted in the HSM_IDS$_1$ (Hardware Security Module 1 of IDS) (see Figure 8-6).
	PK.CI$_{IDS}$.ECDSA	Public key for signature verification in the CI Certificate as defined in section 11.2.1. This public key is a trust anchor for the IDS Certification Paths.
Optional in the IDS, see section 9.1.1	**PK.IDS.ECDSA**	Public key for signature verification in the IDS$_1$ Certificate as defined in section 11.2.6.
	SK.IDS.ECDSA	Private key for signature generation of the ATK.IDS$_1$.ECKA in the HSM_IDS$_1$ (see Figure 8-6).
Optional in the IDS and in the OFL	**PK.IMO.ECDSA**	Public key for signature verification in the IMO Certificate as defined in section 11.2.10.

5.1.2 Short-term Static Keys

The short-term static keys in the IDS are in Table 5-2.

Table 5-2: Short-term Static Keys

Category	Key	Definition
In the IDS	**PK.XCI$_{OFL}$.ECDSA = PK.IDS.ECDSA**	Public key for signature verification in the XCI$_{OFL}$ Certificate as defined in section 11.2.2. This public key is equal to the public key for signature verification in the IDS Certificate as defined in section 11.2.4.

5.2 Ephemeral Keys

The ephemeral keys are generated as defined in section 4.2.1; their mapping and notation are defined in section 4.2.2. Each ephemeral key is linked to an Initialization Vector IV used for integrity checking and an Initialization Vector EIV used for encryption. This triplet is to be understood in the following when the term key is used. These keys shall be destroyed when their session of usage is ended. Table 5-3 defines these ephemeral keys and their assignment:

Table 5-3: Ephemeral Keys

Ephemeral Key Name	Assignment	Description
$K_{CERT\text{-}OFL}$	$KS1_0^3$	Key components for Encryption of the CERT.OFL.ECDSA Certificate
$K_{TRSH\text{-}Ext}$	$KS1_1^3$	Key components for Encryption of the TRSH Extensions
$K_{TIME\text{-}STAMP}$	$KS1_2^3$	Key components for Encryption of the Time-Stamp
K_{SDS}	Initialized by rotation of $KS2_0^3$	Key components for Encryption of the Segment Descriptor Structure
K_{IMD}	$KS2_0^3$	Key components for Encryption of the Image Descriptor (IMD)
K_{ARP}	$KS2_1^3$	Key components for Encryption of the ARP
$K_{IDS\text{-}Ext}$	$KS2_2^3$	Key components for Encryption of the IDS Extension

5.3 Security Protocol

5.3.1 Overview

The security protocol related to the Image Loading has four steps:

- Image Session Data generation
- Image generation
- Image Loading
- Image Session Data deletion

5.3.2 Image Session Data Generation

This step is initiated after receipt of IDS Credentials. The OFL extracts from the IDS Credentials a Certification Path originating from PK.CI$_{OFL}$.ECDSA public key or PK.CI$_{IDS}$.ECDSA and ending at the CERT.IDS$_1$.ECKA Certificate.

This step performs the generation of the ECC ephemeral key pair in the same ECC domain than the public key conveyed by a valid CERT.IDS$_1$.ECKA Certificate.

```
VERIFY_PATH (PK.CI$_{OFL}$.ECDSA)[CERT.IDS$_1$.ECKA]or
VERIFY_PATH (PK.CI$_{IDS}$.ECDSA)[CERT.IDS$_1$.ECKA]
{eSK.OFL.ECKA,ePK.OFL.ECKA}=ECDHE()
```

Where:

PK.CI$_{OFL}$.ECDSA / PK.CI$_{IDS}$.ECDSA	:	The public key originating a Certification Path ending with the CERT.IDS$_1$.ECKA Certificate, as defined in section 11.1.1
CERT.IDS$_1$.ECKA	:	The End Entity Certificate ending the Certification Path
eSK.OFL.ECKA	:	The private ephemeral key for key agreement involved in the computation of all ephemeral keys defined in section 5.2
ePK.OFL.ECKA	:	The public ephemeral key for key agreement involved in the computation of all ephemeral keys defined in section 5.2

Then the following actions are performed:

```
KS1$^9$=DERIVE$^9$(eSK.OFL.ECKA)[PK.IDS$_1$.ECKA]
M$_{CERT-OFL}$, H$_{CERT-OFL}$ =ENCRYPT (K$_{CERT-OFL}$)[CERT.OFL.ECDSA]
M$_{TRSH-Ext}$, H$_{TRSH-Ext}$ =ENCRYPT (K$_{TRSH-Ext}$)[TRSH-Extensions]
ATK.OFL.ECKA=SIGN(SK.OFL.ECDSA)[tbsToken$_{OFL}$]
```

Where:

$M_{CERT\text{-}OFL}$	:	The encryption of CERT.OFL.ECDSA by using the keys $K_{CERT\text{-}OFL}$
$H_{CERT\text{-}OFL}$	:	The integrity check of CERT.OFL.ECDSA by using the keys $K_{CERT\text{-}OFL}$
$M_{TRSH\text{-}Ext}$	:	The encryption of TRSH-Extensions by using the keys $K_{TRSH\text{-}Ext}$
$H_{TRSH\text{-}Ext}$	:	The integrity check of TRSH-Extensions by using the keys $K_{TRSH\text{-}Ext}$
ATK.OFL.ECKA	:	OFL Authentication Token for key agreement
$tbsToken_{OFL}$	:	The body of the Authentication Token as defined in Table 11-14

The TRSH-Extensions are defined in section 11.3.2.

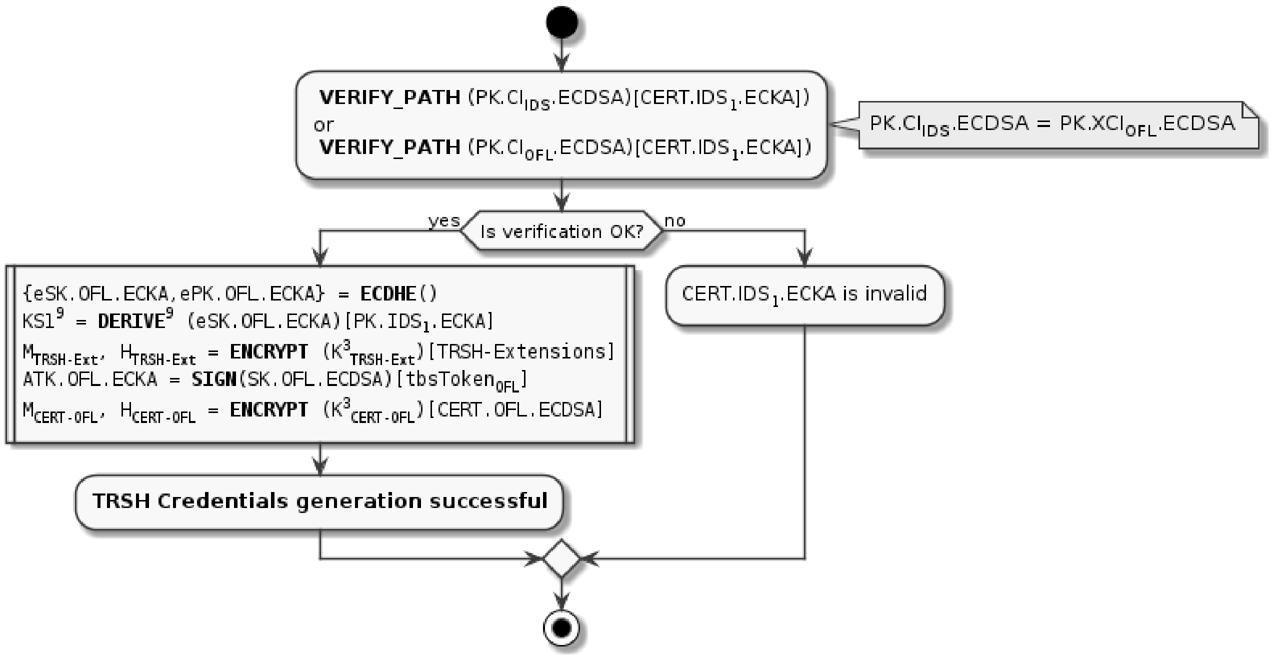
The OFL generates an Image Session Identifier (ISID) as defined in section 2.3.1.2.

The OFL starts the Image Session by storing the Image Session Data identified by ISID, with the following content:

- eSK.OFL.ECKA.
- $K_{TIME\text{-}STAMP}$: The session key needed to decrypt the message $M_{TIME\text{-}STAMP}$ containing the Time-Stamp defined in [ANSI X9.95]
- The shared secret (ShS) used for deriving the key matrix $KS1^9$.

Figure 5-1 shows the preparation and the generation of the TRSH Credentials as described above.

Figure 5-1: Preparation and Generation of the TRSH Credentials



VERIFY_PATH (PK.CI$_{IDS}$.ECDSA)[CERT.IDS$_1$.ECKA])
or
VERIFY_PATH (PK.CI$_{OFL}$.ECDSA)[CERT.IDS$_1$.ECKA])
PK.CI$_{IDS}$.ECDSA = PK.XCI$_{OFL}$.ECDSA
yes
Is verification OK?
no
{eSK.OFL.ECKA, ePK.OFL.ECKA} = ECDHE()
KS1^9 = DERIVE^9 (eSK.OFL.ECKA)[PK.IDS$_1$.ECKA]
M$_{TRSH-Ext}$, H$_{TRSH-Ext}$ = ENCRYPT (K$^3_{TRSH-Ext}$)[TRSH-Extensions]
ATK.OFL.ECKA = SIGN(SK.OFL.ECDSA)[tbsToken$_{OFL}$]
M$_{CERT-OFL}$, H$_{CERT-OFL}$ = ENCRYPT (K$^3_{CERT-OFL}$)[CERT.OFL.ECDSA]
CERT.IDS$_1$.ECKA is invalid
TRSH Credentials generation successful

5.3.3 Image Generation

This part of the Image generation is computed within the HSM_IDS$_1$, using data extracted from the TRSH Credentials to perform the following actions:

```
VERIFY_PATH(PK.CI$_{OFL}$.ECDSA)[CERT.FAC.ECDSA]
KS1[9]=DERIVE[9](SK.IDS$_1$.ECKA)[ePK.OFL.ECKA]
CERT.OFL.ECDSA=DECRYPT(K$_{CERT-OFL}$)[M$_{CERT-OFL}$]
VERIFY(PK.FAC.ECDSA)[CERT.OFL.ECDSA]
VERIFY(PK.OFL.ECDSA)[ATK.OFL.ECKA]
TRSH-Extensions,H$_{TRSH-Ext}$=DECRYPT(K$_{TRSH-Ext}$)[M$_{TRSH-Ext}$]
M$_{TIME-STAMP}$, H$_{TIME-STAMP}$= ENCRYPT (K$_{TIME-STAMP}$)[TIME-STAMP]
ATK.IDS$_1$.ECKA Authentication Token or tbsToken$_{IDS1}$ generation for interfacing
HSM_IDS$_2$.
```

This part of the Image generation is computed within HSM_IDS$_2$, which ensures the following actions:

```
{eSK.IDS$_2$.ECKA, ePK.IDS$_2$.ECKA}=ECDHE()
KS2[9]=DERIVE[9](eSK.IDS$_2$.ECKA)[ePK.OFL.ECKA]
M$_{IDS2-Ext}$, H$_{IDS2-Ext}$ = ENCRYPT (K$_{IDS2-Ext}$)[IDS2-Extensions]
ATK.IDS$_2$.ECKA=SIGN(SK.IDS$_2$.ECDSA)[tbsToken$_{IDS2}$]
M$_{IMD}$, H$_{IMD}$ = ENCRYPT (K$_{IMD}$)[IMD]
For each SDS of the Image from 1 to N
      M$_{SDS-I}$, H$_{SDS-I}$=ENCRYPT(K$_{SDS}$)[SDS$_I$]
If ARP in LOCKED state then
      Lookup CERT.ARP.ECDSA with UUID$_{OFLA}$ from ATK.OFL.ECKA
      M$_{ARP}$, H$_{ARP}$ = ENCRYPT(K$_{ARP}$)[CERT.ARP.ECDSA]
```

The stored Image that is ready for download contains the following data:

M_{IMD}	:	The IMD encrypted by using the key K_{IMD}.
H_{IMD}	:	Integrity check of IMD by using the key K_{IMD}.
M_{SDS-I}	:	The SDS$_I$ encrypted by using the key K_{SDS}.
H_{SDS-I}	:	Integrity check of SDS$_I$ by using the key K_{SDS}.
M_{ARP}	:	CERT.ARP.ECDSA encrypted by using the key K_{ARP}.
H_{ARP}	:	Integrity check of CERT.ARP.ECDSA by using the key K_{ARP}.
$M_{TIME-STAMP}$	:	The time stamp encrypted by using the key $K_{TIME-STAMP}$.
$H_{TIME-STAMP}$	:	Integrity check of time stamp by using the key $K_{TIME-STAMP}$.
ATK.IDS$_2$.ECKA	:	The Authentication Token generated by HSM_IDS$_2$ during the Image binding process.
CERT.IDS$_2$.ECDSA	:	The IDS$_2$ Certificate.
CERT.CI$_{OFL}$.ECDSA	:	The CI$_{OFL}$ Certificate.
MANIFEST	:	Contains the OFL Agent instructions for the Agent loading the Image into the OFL
IMAGE	:	The Bound Image to load into the OFL

Figure 5-2 shows the TRSH Credentials verification as described above.

Figure 5-2: TRSH Credentials Verification

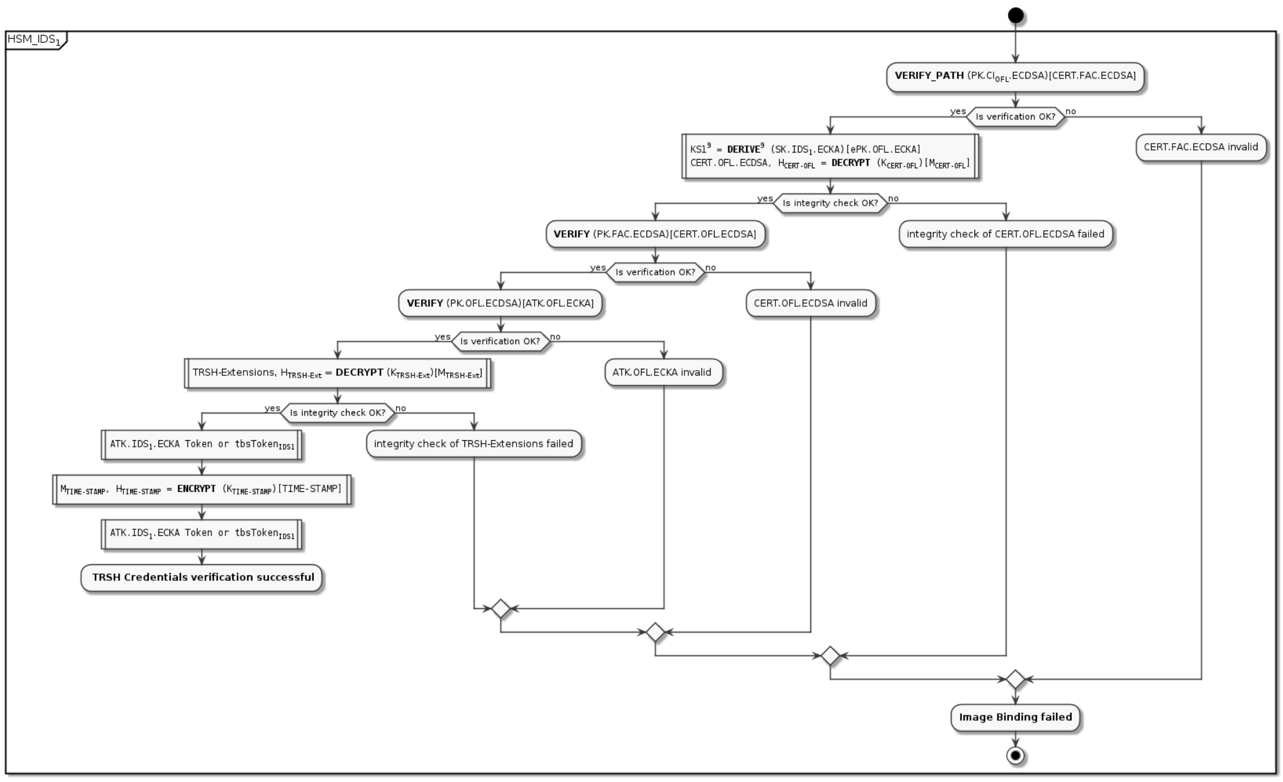
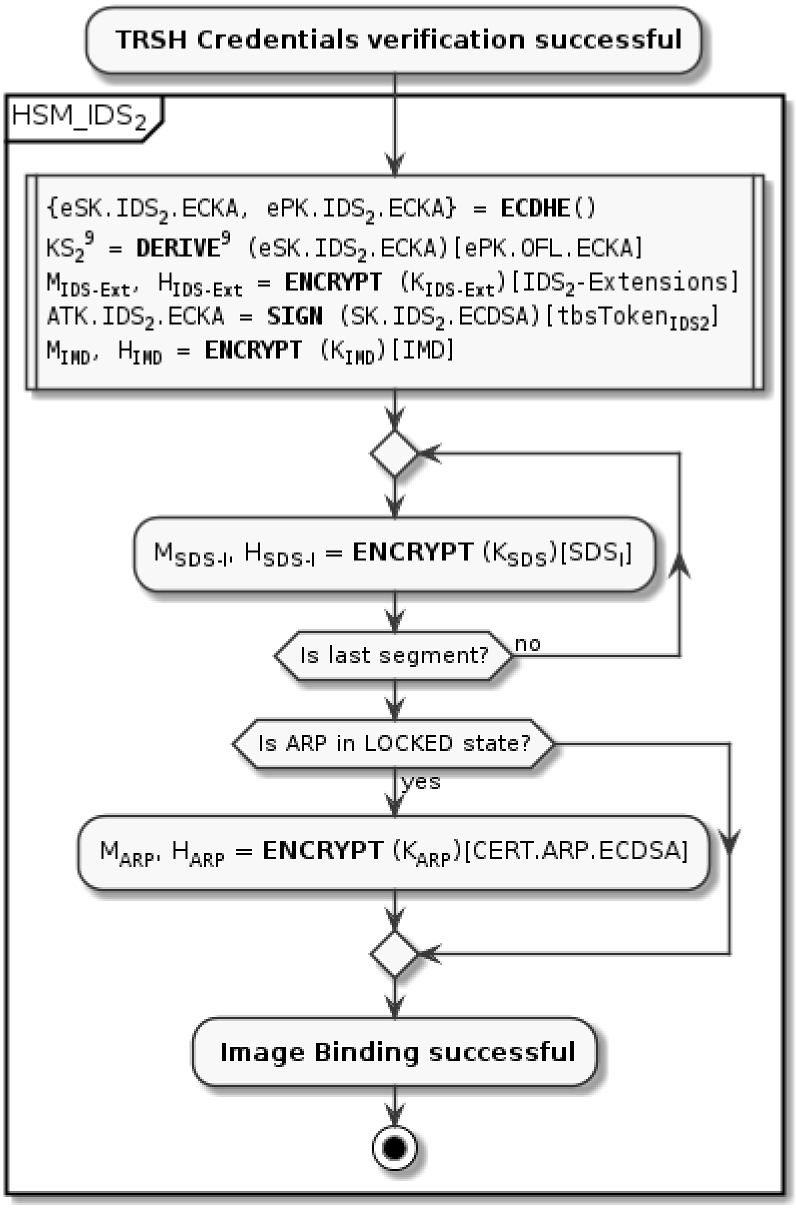

HSM_IDS1
VERIFY_PATH (PK.CIOFL.ECDSA)[CERT.FAC.ECDSA]
yes
Is verification OK?
no
KS1⁹ = DERIVE⁹ (SK.IDS1.ECKA)[ePK.OFL.ECKA]
CERT.OFL.ECDSA, HCERT-OFL = DECRYPT (KCERT-OFL)[MCERT-OFL]
CERT.FAC.ECDSA invalid
yes
Is integrity check OK?
no
VERIFY (PK.FAC.ECDSA)[CERT.OFL.ECDSA]
integrity check of CERT.OFL.ECDSA failed
yes
Is verification OK?
no
VERIFY (PK.OFL.ECDSA)[ATK.OFL.ECKA]
CERT.OFL.ECDSA invalid
yes
Is verification OK?
no
TRSH-Extensions, HTRSH-Ext = DECRYPT (KTRSH-Ext)[MTRSH-Ext]
ATK.OFL.ECKA invalid
yes
Is integrity check OK?
no
ATK.IDS1.ECKA Token or tbsTokenIDS1
integrity check of TRSH-Extensions failed
MTIME-STAMP, HTIME-STAMP = ENCRYPT (KTIME-STAMP)[TIME-STAMP]
ATK.IDS1.ECKA Token or tbsTokenIDS1
TRSH Credentials verification successful
Image Binding failed

Figure 5-3 shows the Image Binding generation as described above.

Figure 5-3: Image Binding Generation


TRSH Credentials verification successful
HSM_IDS_2
$\{eSK.IDS_2.ECKA, ePK.IDS_2.ECKA\}$ = **ECDHE**()
KS_2^9 = **DERIVE**^{9} $(eSK.IDS_2.ECKA)[ePK.OFL.ECKA]$
$M_{IDS\text{-}Ext}, H_{IDS\text{-}Ext}$ = **ENCRYPT** $(K_{IDS\text{-}Ext})[IDS_2\text{-}Extensions]$
$ATK.IDS_2.ECKA$ = **SIGN** $(SK.IDS_2.ECDSA)[tbsToken_{IDS2}]$
M_{IMD}, H_{IMD} = **ENCRYPT** $(K_{IMD})[IMD]$
$M_{SDS\text{-}I}, H_{SDS\text{-}I}$ = **ENCRYPT** $(K_{SDS})[SDS_I]$
Is last segment?
no
Is ARP in LOCKED state?
yes
M_{ARP}, H_{ARP} = **ENCRYPT** $(K_{ARP})[CERT.ARP.ECDSA]$
Image Binding successful

5.3.4 Image Loading

This operation is computed by the OFL and performs the following actions:

```
If $M_{TIME-STAMP}$ and $H_{TIME-STAMP}$ are present then
      TIME-STAMP, $H_{TIME-STAMP}$ = DECRYPT($K_{TIME-STAMP}$)[$M_{TIME-STAMP}$]
Check time window validity of the IDS Certificate versus TIME-STAMP
VERIF_PATH(PK.$CI_{OFL}$.ECDSA)[CERT.$IDS_2$.ECDSA]
VERIFY(PK.$IDS_2$.ECDSA)[ATK.$IDS_2$.ECKA]
KS2[9]=DERIVE[9](eSK.OFL.ECKA)[ePK.$IDS_2$.ECKA]

CERT.ARP.ECDSA,$H_{ARP}$=DECRYPT($K_{ARP}$)[$M_{ARP}$]
VERIFY(PK.$IMO_{OFLA}$.ECDSA) [CERT.ARP.ECDSA]
or
VERIFY(PK.$TRSH_M$.ECDSA) [CERT.ARP.ECDSA]
Check time window validity of the ARP versus TIME-STAMP
IDS2-Extensions, $H_{IDS2-Ext}$=DECRYPT($K_{IDS-Ext}$)[$M_{IDS2-Ext}$]
IMD,$H_{IMD}$=DECRYPT($K_{IMD}$)[$M_{IMD}$]
Check validity of the OFL Operations versus ARP
For each $SDS_I$ for I=1 to N
      $SDS_I$, $H_{SDS-I}$=DECRYPT($K_{SDS}$)[$M_{SDS-I}$]
      $SEG_I$, $H_{SEG-I}$=DECRYPT($K_{SEG}$)[$M_{SEG-I}$]
      Load in the TRSH the Firmware Segment I
```

The first time a Firmware is loaded is referred to as the Initial Firmware Loading.

Initial Firmware Loading is possible if no existing Firmware Session contains a private Firmware Identifier equal to the one within the IMD of the loaded Image. (See section 7.4.1.1 for details of IMD format.)

Upon successful processing of Initial Firmware Loading, the OFL creates a Firmware Session described by its Firmware Session Data (see section 8.5).

Firmware Update processing shall abort if the current version of the Firmware (VER_CURR) in the Image Descriptor (IMD) and in the Firmware Session Data are not equal.

Upon successful completion of Firmware Update processing, the current version VER_CURR shall be replaced with the VER_NEXT.

Figure 5-4 shows the time stamp (TIME-STAMP) and IDS_2 Certificate validation as detailed above.

Figure 5-4: Time Stamp and IDS_2 Certificate Verification

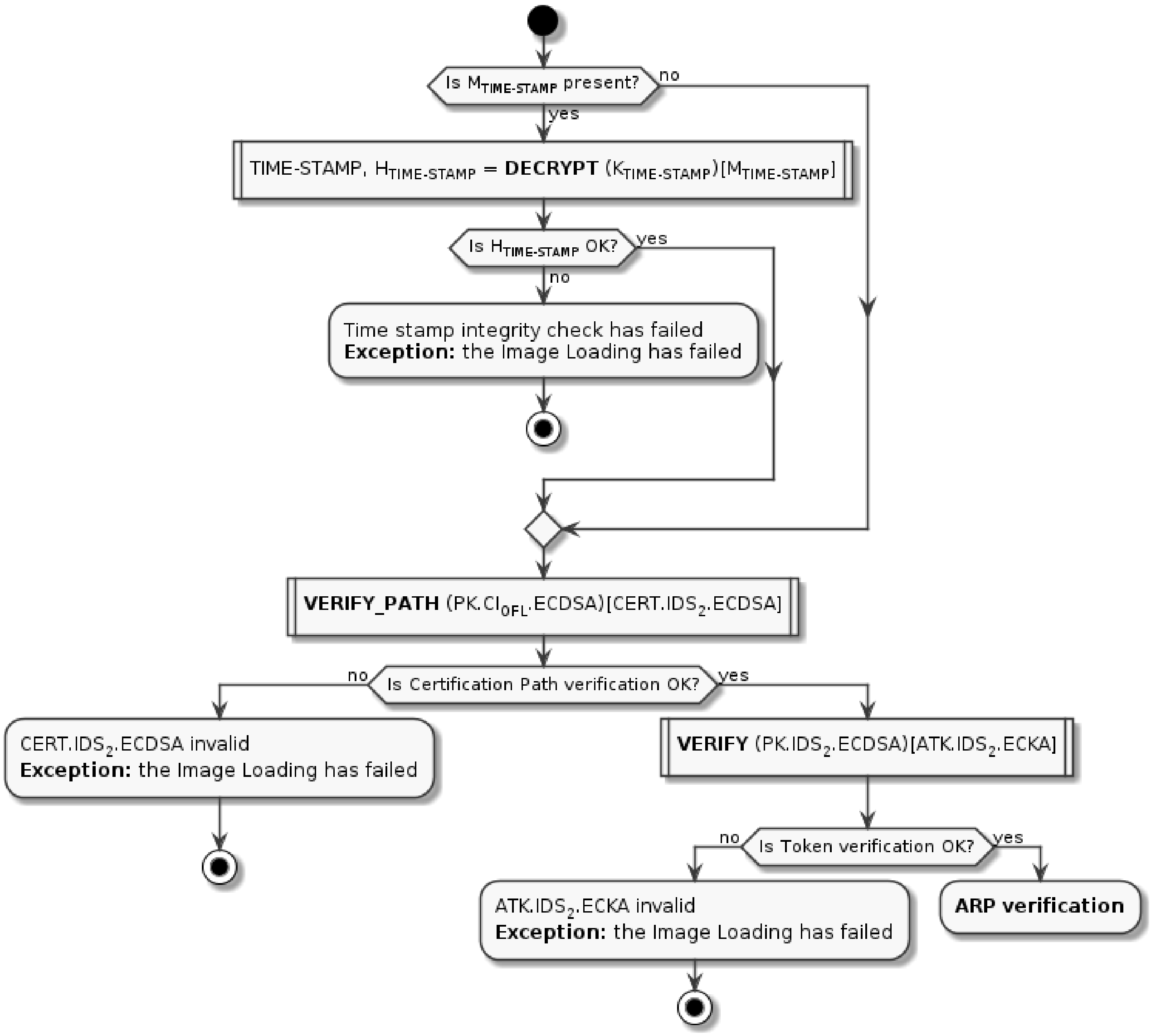

Is $M_{TIME\text{-}STAMP}$ present?
no
yes
TIME-STAMP, $H_{TIME\text{-}STAMP}$ = **DECRYPT** ($K_{TIME\text{-}STAMP}$)[$M_{TIME\text{-}STAMP}$]
Is $H_{TIME\text{-}STAMP}$ OK?
yes
no
Time stamp integrity check has failed
Exception: the Image Loading has failed
VERIFY_PATH (PK.CI_{OFL}.ECDSA)[CERT.IDS_2.ECDSA]
no
Is Certification Path verification OK?
yes
CERT.IDS_2.ECDSA invalid
Exception: the Image Loading has failed
VERIFY (PK.IDS_2.ECDSA)[ATK.IDS_2.ECKA]
no
Is Token verification OK?
yes
ATK.IDS_2.ECKA invalid
Exception: the Image Loading has failed
ARP verification

Figure 5-5 illustrates the ARP verification if the ARP is in the LOCKED state.

Figure 5-5: ARP Verification

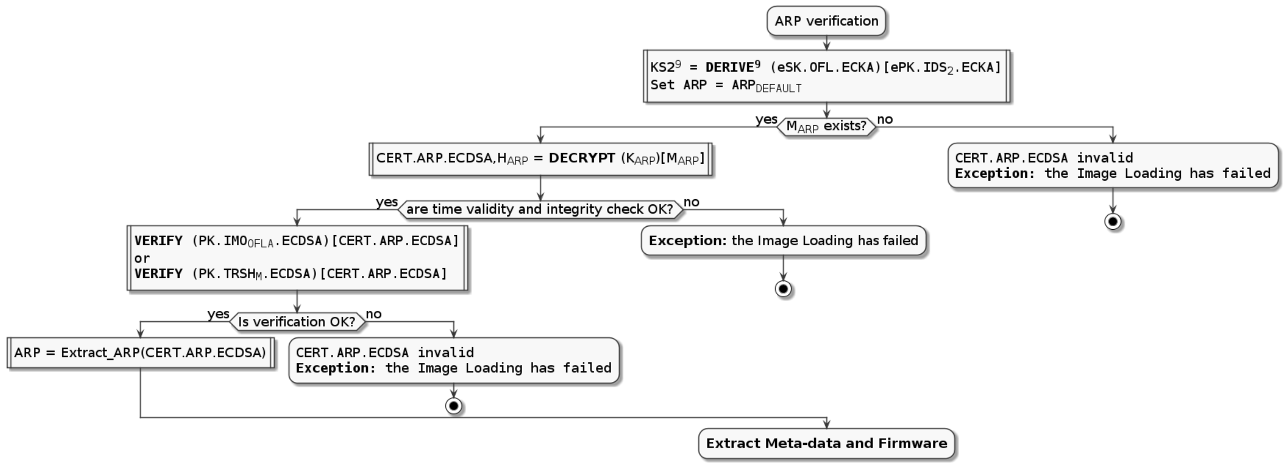
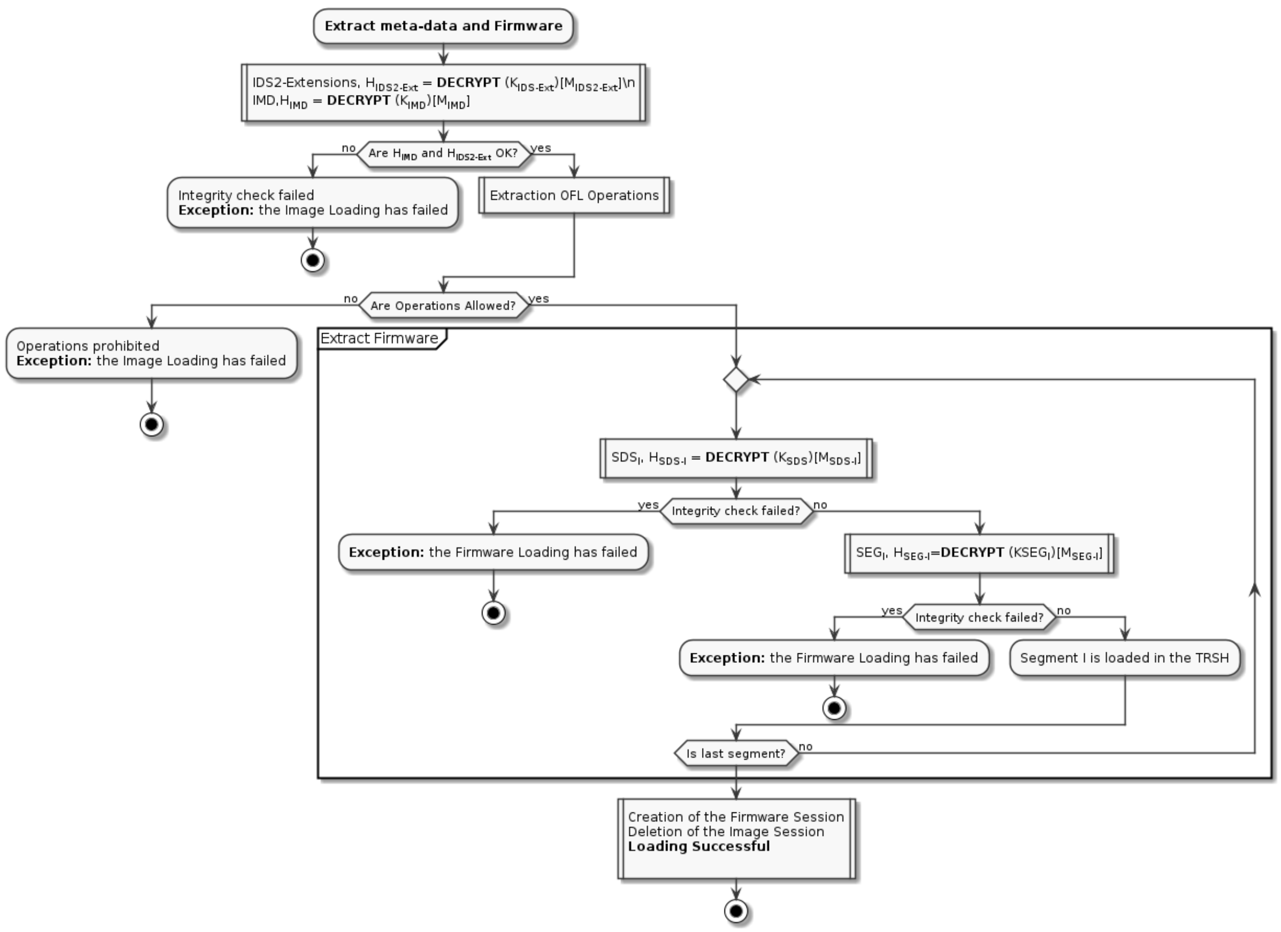

ARP verification
KS2[9] = DERIVE[9] (eSK.OFL.ECKA)[ePK.IDS2.ECKA]
Set ARP = ARPDEFAULT
MARP exists?
yes
no
CERT.ARP.ECDSA,HARP = DECRYPT (KARP)[MARP]
CERT.ARP.ECDSA invalid
Exception: the Image Loading has failed
are time validity and integrity check OK?
yes
no
VERIFY (PK.IMOOFLA.ECDSA)[CERT.ARP.ECDSA]
or
VERIFY (PK.TRSHM.ECDSA)[CERT.ARP.ECDSA]
Exception: the Image Loading has failed
Is verification OK?
yes
no
ARP = Extract_ARP(CERT.ARP.ECDSA)
CERT.ARP.ECDSA invalid
Exception: the Image Loading has failed
Extract Meta-data and Firmware

Figure 5-6 illustrates the extraction of the IDS metadata and the Firmware as detailed above.

Figure 5-6: Extraction of the IDS Metadata and the Firmware


Extract meta-data and Firmware
IDS2-Extensions, HIDS2-Ext = DECRYPT (KIDS-Ext)[MIDS2-Ext]\n
IMD,HIMD = DECRYPT (KIMD)[MIMD]
Are HIMD and HIDS2-Ext OK?
no
yes
Integrity check failed
Exception: the Image Loading has failed
Extraction OFL Operations
Are Operations Allowed?
no
yes
Operations prohibited
Exception: the Image Loading has failed
Extract Firmware
SDSI, HSDS-I = DECRYPT (KSDS)[MSDS-I]
Integrity check failed?
yes
no
Exception: the Firmware Loading has failed
SEGI, HSEG-I=DECRYPT (KSEGI)[MSEG-I]
Integrity check failed?
yes
no
Exception: the Firmware Loading has failed
Segment I is loaded in the TRSH
Is last segment?
no
Creation of the Firmware Session
Deletion of the Image Session
Loading Successful

5.3.5 Image Session Data Deletion

The OFL discards the Image Session Data and stops the Image Session when:

- The Image Loading is successful, or
- the Image Loading is unsuccessful, or
- the number of concurrent Image Sessions exceeds a given limit (implementation dependent) due to a request for a new Image Session which is the oldest generated.

Annex D describes an overview of the whole security protocol from Image Session Data Generation through Image Session Data Deletion.

5.4 Security Properties

5.4.1 Mutual Authentication

Authentication is achieved by performing the following authentication steps. This ensures that the OFL and the components of IDS are authentic by means of verification of their respective Certification Paths, as defined in section 11.1.2:

- The OFL validates HSM_IDS$_1$ as a trust part of IDS by verifying the IDS Certification Path ending with CERT.IDS$_1$.ECKA and originated by one of two possible public keys as trust anchors:
  - PK.CI$_{OFL}$.ECDSA by using a Certification Path including the CERT.XCI$_{OFL}$.ECDSA Certificate or
  - PK.CI$_{IDS}$.ECDSA by using a Certification Path including the CERT.IDS.ECDSA Certificate.

  The OFL authenticates the valid HSM_IDS$_1$ by generating ephemeral session keys K$_{CERT\text{-}OFL}$ in order to generate the cryptogram M$_{CERT\text{-}OFL}$, H$_{CERT\text{-}OFL}$. Only an authentic HSM_IDS$_1$ can derive K$_{CERT\text{-}OFL}$, decrypt M$_{CERT\text{-}OFL}$, and check the integrity of the CERT.OFL.ECDSA by using H$_{CERT\text{-}OFL}$.
- As a trust part of IDS, HSM_IDS1 authenticates the OFL by:
  - Validating the Certification Path originated by the PK.CI$_{OFL}$.ECDSA public key and ending with the CERT.OFL.ECDSA Certificate.
  - Verifying the signature of the ATK.OFL.ECKA Authentication Token using the PK.OFL.ECDSA public key.
  - Challenging for authentication the OFL by using CODE_M and/or IDS-CHALLENGE as a challenge(s) as defined in section 5.4.2.
- The OFL validates HSM_IDS$_2$ as a trust part of IDS by verifying the IDS Certification Path ending with CERT.IDS$_2$.ECDSA and originated by one of two possible public keys as trust anchors:
  - PK.CI$_{OFL}$.ECDSA by using a Certification Path including the CERT.XCI$_{OFL}$.ECDSA Certificate or
  - PK.CI$_{IDS}$.ECDSA by using a Certification Path including the CERT.IDS.ECDSA Certificate.
- The OFL authenticates the valid HSM_IDS2 by:
  - Verifying the signature of the ATK.IDS$_2$.ECKA Authentication Token using the PK.IDS$_2$.ECDSA public key.
  - Challenging for authentication of the IDS by using the ISID of the current Image Session as a challenge.

5.4.2 Anti-replay

Anti-Replay is available in OFL for loading operation (unicast) only. The ISID and the ephemeral session keys {eSK.OFL.ECKA, ePK.OFL.ECKA}, KS2^9 key matrix, and KS1^9 key matrix are only valid in the scope of a single Image Session.

CODE_M may be discarded after use in order to prevent DoS and replay attacks in IDS. CODE_M is defined in section 6.2.2.

IDS-CHALLENGE may be discarded after use and is provided by the IDS (or by any other source); may be used to prevent DoS and replay attacks in IDS. IDS-CHALLENGE is defined in section 6.2.2.

5.4.3 Perfect Forward Secrecy

Perfect Forward Secrecy is achieved by using two ephemeral key pairs {eSK.OFL.ECKA, ePK.OFL.ECKA} and {eSK.IDS$_2$.ECKA, ePK.IDS$_2$.ECKA} for computing KS1[9] key matrix and KS2[9] key matrix. These ephemeral key pairs shall be destroyed at the end of the Image Session.

This property is only valid for loading operation (unicast).

5.4.4 Integrity and Confidentiality

The integrity and confidentiality of the data transfer are ensured by using the eGCM-AES-128 or eGCM-AES-256 in order to check the integrity and perform the encryption/decryption of data in a single pass. Both algorithms shall perform a key rotation of 128 bits or 256 bits every 16 blocks.

5.4.5 OFL Operation Tokens

5.4.5.1 Initiated from the IDS

Subsequent to a successful Image Loading, OFL shall generate an Operation Token readable from the OFL registry via the OPERATION_TOKEN registry entry.

The Operation Token is computed as follows:

- ATK_OFL_DO_OPERATE has a 256-bit Operation Token: ATK_OFL_DO_OPERATE = SHA-256($UUID_P$ |$UUID_I$| '10'). This token is only available for unicast Image Loading.

Where:

$UUID_P$	:	Private Identifier of the Firmware read from the IMD (Image Descriptor) as defined in section 7.4.1.1
$UUID_I$	:	Public Identifier of the Firmware read from the IMD as defined in section 7.4.1.1

5.4.5.2 Initiated from the OFL Agent

Subsequent to a change to a Firmware state (see section 8.4.3) not due to an Image Loading, the OFL Operation Token shall be generated as follows:

- ATK_OFL_STATE_ENABLED has a 256-bit Operation Token:
  - ATK_OFL_ STATE_ENABLED_SEED = SHA-256($UUID_P$ | $UUID_I$ |'11').
  - ATK_OFL_ STATE_ENABLED = SHA-256(ATK_OFL_STATE_ENABLED_SEED|Counter).
- ATK_OFL_STATE_DISABLED has a 256-bit Operation Token:
  - ATK_OFL_ STATE_DISABLED_SEED = SHA-256($UUID_P$ | $UUID_I$ |'12').
  - ATK_OFL_ STATE_DISABLED = SHA-256(ATK_OFL_STATE_DISABLED_SEED|Counter).

The ATK_OFL_ STATE_ENABLED_SEED and ATK_OFL_ STATE_DISABLED_SEED shall only be known by OFL and the IDS.

Where:

$UUID_P$	:	Private Identifier of the Firmware read from the Firmware Session
$UUID_I$	:	Public Identifier of the Firmware read from the Firmware Session
Counter	:	A 32-bit counter which is set to 0 at the creation of the Firmware Session and shall be incremented after each change of the Firmware state not due to Image Loading

The OFL Agent commands requesting the change of the Firmware state as defined in section 8.4.3 are out of scope of this specification.

The interface for transferring the ATK_OFL_STATE_ENABLED_SEED and ATK_OFL_STATE_DISABLED _SEED values for collecting the Operation Tokens in order to allow the Image Owner (IMO) to process the Operation Token, is out of scope of this document.

5.4.5.3 Initiated from the IDS or from the OFL Agent

The deletion of a Firmware may be subsequent to one of following actions:

- An Image Loading requiring a Delete Firmware operation as defined in section 8.6.
- A request from the OFL Agent requiring the deletion of a Firmware.

The above actions shall generate an Operation Token readable from the OFL registry via the OPERATION_TOKEN registry entry.

This Operation Token is computed as follows:

- ATK_OFL_DELETE_SESSION has a 256-bit Operation Token: ATK_OFL_DELETE_SESSION = SHA-256(UUID$_P$ | UUID$_I$ |'15')

Where:

UUID$_P$	:	Private Identifier of the Firmware read from the IMD as defined in section 7.4.1.1
UUID$_I$	:	Public Identifier of the Firmware read from the IMD as defined in section 7.4.1.1

The OFL Agent commands requesting the deletion of the Firmware is out of scope of this specification.

5.5 GDPR Support

There are no data exchanges in plaintext involving large constants of the OFL such as the ones included in a Certificate or an identifier and any part of the IDS (even internally), except within the HSM_IDS$_1$. There is no way for any Image Owner sharing a Secure Hardware Platform to correlate data between two different Images. There is no way for an observer of the data traffic between the OFL and the IDS to correlate data and to derive the Secure Hardware Platform identity (SN) of the TRSH without involving the TRSH Maker.

There is no data exchange in plaintext involving large constants of the OFL and the OFL Agent such as:

- The identification of the Secure Hardware Platform
- The diversified static keys
- The serial number and the batch number

The Privacy Protection is ensured by:

- Pseudonymisation of the serial number (SN) called SERIAL_NUMBER of the Secure Hardware Platform.
- Pseudonymisation of the batch number (BN) called BATCH_NUMBER of the Secure Hardware Platform.
- CERT.OFL.ECDSA Certificate is encrypted by an ephemeral key $K_{CERT\text{-}OFL}$ which is only known by HSM_IDS$_1$ and therefore not accessible by the Image Maker in charge to generate an Image.

No OFL static key pairs are used to ensure the integrity and confidentiality of data to and from the OFL.

6 Exchanged Parameters from the OFL Agent Perspective

6.1 Format

The Image format from the OFL Agent perspective is defined in Figure 3-3.

6.2 Parameters

6.2.1 Data Elements

The following data elements are used in the parameters related to the OFL data flow defined in section 9.1.

```
-- ASN1START
OFL-Token-20-Schema { iso(1) member-body(2) us(840) globalplatform(114283)} DEFINITIONS
EXPLICIT TAGS
EXTENSIBILITY IMPLIED
::= BEGIN
EXPORTS ALL; --
IMPORTS
Certificate,                          --RFC 5280 Certificate X.509v3
id-pkix,
Extensions,                           --RFC 5280 X.509v3 extension
Extension,
ECDSA-Sig-Value,
AlgorithmIdentifier,
Attribute,
AttributeType,
AttributeValue,
AttributeTypeAndValue,
SubjectPublicKeyInfo,
UniqueIdentifier,
Validity
FROM PKIX1Explicit88;

Version                               ::= INTEGER  {  v1(0)  }
CertificationPath                     ::= SET OF Certificate
Cryptogram                            ::= OCTET STRING
IntegrityCheck                        ::= BIT STRING

UUID                                  ::= OCTET STRING (SIZE(16))
UUIDF                                 ::= UUID
                                      -- Firmware Familly Identifier
EncryptedBlock                        ::= [PRIVATE 8] SEQUENCE
{
aM                                    Cryptogram,
                                      --Encrypted message
aH                                    IntegrityCheck
                                      -- Integrity check
}
ATK-Content ::=[PRIVATE 9] CHOICE
{
aATK-OFL-ECKA                         ATK-OFL-ECKA-Content,
aATK-IDS2-ECKA                        ATK-IDS2-ECKA-Content,
aATK-IDS1-ECKA                        ATK-IDS1-ECKA-Content
}
Token  ::=  SEQUENCE
{
tbsToken                              TBSToken,
signatureAlgorithm                    AlgorithmIdentifier,
signature                             ECDSA-Sig-Value
}

TBSToken  ::=  SEQUENCE
{
version                               [0]  Version DEFAULT v1,
subjectPublicKeyInfo                  SubjectPublicKeyInfo,
aATK-Content                          ATK-Content,
extensions                              [8]  Extensions OPTIONAL
}
-- ASN1STOP
```

6.2.2 IDS Credentials Parameter

6.2.2.1 Description

A set of IDS Credentials Parameter is used for the generation of the TRSH Credentials Parameter.

The following ASN.1 description defines the IDS Credentials Parameter (IDS_CREDENTIALS_PARAMETER).

```
-- ASN1START
IDS_CREDENTIALS_PARAMETER ::= [APPLICATION 1] SEQUENCE
{
aIDS1-CertificationPath                 [0]CertificationPath,
                                        --Certification Path to the CERT.IDS1.ECKA Certificate
aIDS-Revocation-Status                  [1] OCTET STRING OPTIONAL,
aIDS1-Extensions                        Extensions --extensions
}
id-ofl-ids-parameter OBJECT IDENTIFIER ::= {id-globalplatform ofl(10) ids-parameter(4)}
id-ofl-ids-parameter-ids-challenge OBJECT IDENTIFIER ::= {id-ofl-ids-parameter ids-challenge(1)}
id-ofl-ids-parameter-code-m OBJECT IDENTIFIER ::= {id-ofl-ids-parameter code-m(2)}
id-ofl-ids-parameter-urn OBJECT IDENTIFIER ::= {id-ofl-ids-parameter urn(3)}

-- ASN1STOP
```

Where:

aIDS1-CertificationPath	:	List of the Certificates of the IDS1 Certification Path originating from one of the following: • PK.CI$_{OFL}$.ECDSA public key ended by the IDS$_1$ Certificate (CERT.IDS$_1$.ECKA) for key agreement; or • PK.CI$_{IDS}$.ECDSA public key ended by the IDS$_1$ Certificate (CERT.IDS$_1$.ECKA) for key agreement
aIDS-Revocation-Status	:	List of the Online Certificate Status Protocol (OCSP) responses as defined in section 11.1.3.4.3
aIDS1-Extensions	:	Parameter from the IDS

6.2.2.2 Extensions

The list of extensions in IDS1-Extensions shall contain the extension described by the following parameters:

extnID	:	id-ofl-ids-parameter-code-m
critical	:	true
extnValue	:	CODE_M (16 bytes) as the Matching Code allowing retrieving a pending Image in the IDS

```
-- ASN1START
CODE_M                                  ::= [PRIVATE 10]UUID
                                        --Matching Code
-- ASN1STOP
```

The list of extensions in IDS1-Extensions may contain the extension described by the following parameters:

extnID	:	id-ofl-ids-parameter-ids-challenge
critical	:	false
extnValue	:	IDS-CHALLENGE (4 to 16 bytes) as the challenge allowing the authentication of the OFL and retrieving a pending Image in the IDS when CODE_M is provided by the OFL Agent by using an SHPPI as an identifier

```
-- ASN1START
IDS-CHALLENGE                                 ::= [PRIVATE 11]OCTET STRING (SIZE(4..16))
                                              --Challenges in Additional IDS Credentials
-- ASN1STOP
```

The list of extensions in IDS1-Extensions may contain the extension described by the following parameters:

extnID	:	id-ofl-ids-parameter-urn
critical	:	false
extnValue	:	URN provided by the IDS and used to generate a Deterministic SHPPI (DET_PI), as defined in section 10.2

```
-- ASN1START
URN                                           ::= IA5String (SIZE(1..49))
                                              --URN provided by the IDS for generating a deterministic SHPPI
-- ASN1STOP
```

6.2.2.3 Organization Dependent Extensions

This section is intentionally blank and aims at informing other organizations that they may extend the IDS Credentials extensions for specific purposes.

6.2.3 TRSH Credentials Parameter

OFL shall manage the Image Session identified by ISID in a persistent Image Session memory area. The implementation shall put in place a **circular buffer of at least eight sessions** in such a way that if all Image Sessions are in the pending state, then the oldest one is overwritten by the newest Image Session.

The TRSH Credentials Parameter shall not be parsed by the OFL Agent. This parameter corresponds to the TRSH Credentials (TRSH_CREDENTIALS_PARAMETER) defined in section 7.3.

There is one Image Session per TRSH Credentials Parameter.

6.3 Image Format

6.3.1 OFL_DO_OPERATE Parameter

This parameter contains:

- The Binder Descriptor as defined in section 7.4.2
- The Image Descriptor as defined in section 7.4.1.1

6.3.2 OFL_CHANGE_SEGMENT Parameter

This parameter contains the Segment Descriptor Structure (SDS) for an Image Segment as defined in section 7.4.1.2.

6.3.3 OFL_LOAD_SEGMENT Parameter

This parameter contains an Image Segment as defined in section 7.4.1.3.

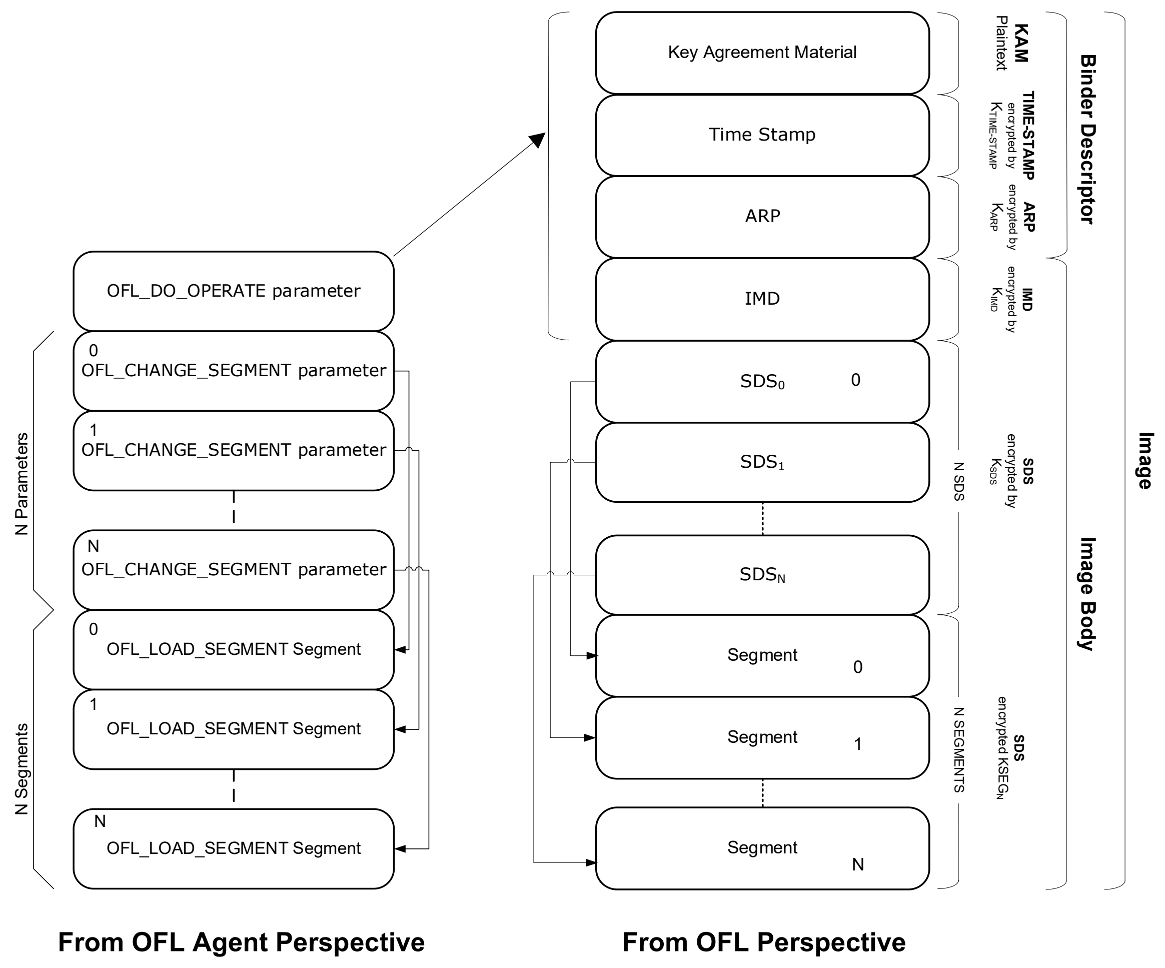
7 Image Format from the OFL Perspective

7.1 Overall Image Format

The format of the Image is viewed from different perspectives depending on the entity that is handling the Image.

Figure 7-1 illustrates the Image format perspectives.

Figure 7-1: Image Format Perspectives



The same Image has two views:

- The view from the OFL Agent perspective
- The view from the OFL perspective

Table 7-1 illustrates the mapping of the two views.

Table 7-1: OFL Command Parameters

OFL Agent View	OFL View
OFL_DO_OPERATE command parameter	OFL_DO_OPERATE command parameter as defined in section 6.3.1
OFL_CHANGE_SEGMENT command parameter	Segment Descriptor Structure (SDS) as defined in section 7.4.1.2
OFL_LOAD_SEGMENT command parameter	Image Segment as defined in section 7.4.1.3

To allow the two views, the following process is necessary.

The Firmware Maker provides the Firmware Segments encrypted by using K_{SEG} and generates the encrypted Security Scheme Descriptor (SSD), which contains the IMD and the list of all the SDS including the K_{SEG} as defined in Figure 7-2. K_{SEG} should be diversified per Firmware Segment. How K_{SEG} is generated by the Firmware Maker is out of scope of this document.

The Image Maker manages the encrypted Firmware Segments and the associated encrypted SSD in order to combine them to generate on-demand unbound Images.

The means for sharing K_{SEG}, managing the encrypted SDS, and managing the intermediate repository to store the encrypted segments from the Firmware Maker are out of scope of this document.

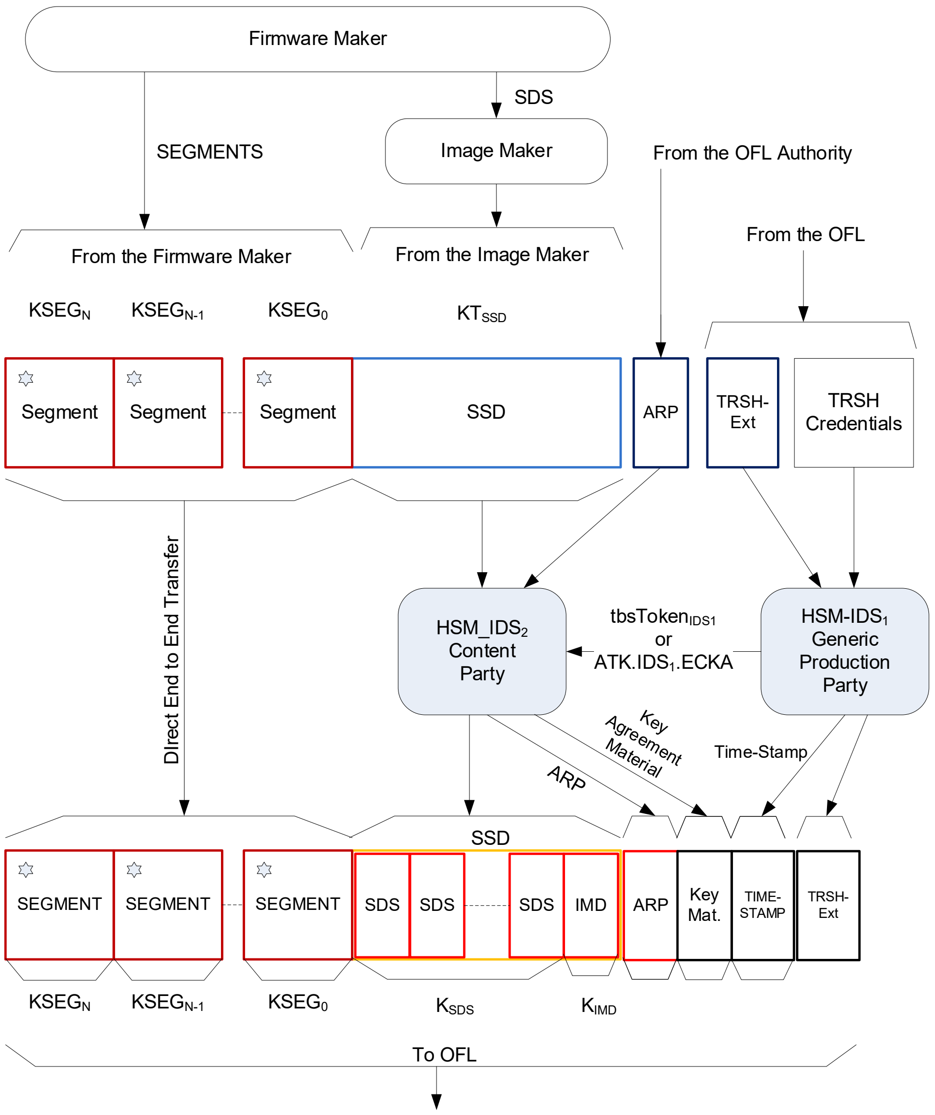
The interface between the Firmware Maker and the Image Maker is out of scope of this document.

The process related to the IDS has three steps:

- The HSM_IDS_1 generates the key agreement material from the IDS Credentials and the TRSH Credentials.
- The HSM_IDS_2 encrypts the ARP provided by the Image Owner.
- The HSM_IDS_2 re-encrypts the SSD from the Image Maker selected by the Image Owner.

Figure 7-2 illustrates the process above involving the Firmware Maker, the Image Maker, and the IDS and resulting in the OFL view of the Image Format.

Figure 7-2: Image Format Process



7.2 Data Elements

The following data elements are used in the parameters related to the OFL data flow defined in section 9.1.

```
-- ASN1START

CODE_M                                ::= [PRIVATE 12]UUID
                                      --Matching Code of a pending Image in the IDS
OFLVersion                            ::= [PRIVATE 13]OCTET STRING (SIZE(2))
                                      --Major (MM) and Minor (mm) version of the OFL
IsPseudonym                           ::= [PRIVATE 14]BOOLEAN
                                      --Boolean is True (1) if CODE_M is an SHPPI of the TRSH
-- ASN1STOP
```

7.3 TRSH Credentials

The following ASN.1 description defines the TRSH Credentials. This TRSH Credentials corresponds to the TRSH Credentials Parameter defined in section 6.2.3.

```
-- ASN1START
TRSH_CREDENTIALS_PARAMETER ::= [APPLICATION 2] SEQUENCE
{
aOFL-Token                              Token,
                                        --ATK.OFL.ECKA Token, includes encrypted TRSH-Extensions
aPN-CertificationPath                   [0]CertificationPath,
                                        --FAC Certificates
aEncryption-Type                        Encryption-Type,
                                        --Type of symetrical Encryption used for aM-CERT-OFL
aM-CERT-OFL                             EncryptedBlock
                                        --M-CERT-OFL message in OFL
}

-- ASN1STOP
```

Where:

aOFL-Token	:	An ATK.OFL.ECKA Authentication Token; a Token as defined in section 6.2.1 that contains in its aATK-Content an ATK-OFL-ECKA
aPN-CertificationPath	:	List of the Certificates: OFL root Certificate (CERT.CI$_{OFL}$.ECDSA), PN Certificate (CERT.PN.ECDSA), and FAC Certificate (CERT.FAC.ECDSA) as defined in section 11.1.2.3
aEncryption-Type	:	The encryption used for aM-CERT-OFL. Shall be set to eGCM-AES-128 or eGCM-AES-256.
aM-CERT-OFL	:	Contains the cryptogram and the integrity check of the CERT.OFL.ECDSA obtained using eGCM-AES-128 or eGCM-AES-256 as defined in Annex B, with the key K$_{CERT-OFL}$

7.4 Image Format

7.4.1 Image Body

7.4.1.1 Image Descriptor

Table 7-2 lists the items of the IMD (Image Descriptor) format encrypted by the transport key KT_{SSD} then re-encrypted by HSM_IDS_2 with the key K_{IMD}. The algorithm used to encrypt the IMD is indicated in the encryption type of the Image Binder Descriptor in section 7.4.2.

Table 7-2: IMD Format

Name	Parameters	Length in Bytes	OFL release
$UUID_P$	Private Identifier of the Firmware	16	Greater than or equal to v1.3.0
$UUID_I$	Public Identifier of the Firmware	16	
$UUID_F$	Identifier of the Firmware Family	16	
$UUID_M$	Identifier of the Firmware Maker	16	
OP_{IMA}	Requested operations bitmap	4	
NUM_SEG	Number of segments (big endian)	4	
ID_AUTH_G	ISID verification requested	1	
PADDING	RFU	3	
VER_CURR	Current version of the Firmware	2	
VER_NEXT	New version of the Firmware after a successful loading	2	
$UUID_G$	Firmware Group Identifier	16	Greater than or equal to v2.0.0
SHS_G	The shared secret of the Firmware Group allowing the computation of KS3[9].	128	

If the Image Loading is successful, then a Firmware Session is created if the parameter NUM_SEG is greater than 0.

The Firmware Session data defined in section 8.5 shall use these parameters at its creation:

The public Firmware Identifier ($UUID_I$), private Firmware Identifier ($UUID_P$), and Firmware Maker Identifier ($UUID_M$) stored in the Firmware Session data shall be immutable once the Image Loading process is initiated.

The following parameters shall be updated in the Firmware Session data for each Firmware Update:

$UUID_F$	:	The public Identifier of the Firmware Family defined by the Firmware Maker.
$LOCAL\_ARP_N$	:	Deduce from OP_{IMA} Requested operations 32-bit bitmap, as defined in section 7.4.1.1 and according to the procedure defined in section 9.4.
VER_NEXT	:	Shall be stored in the Firmware Session as VER_CURR after successful loading. The values of VER_NEXT and VER_CURR and how they are assigned are out of scope of the present document and in the responsibility of the Firmware Maker.
SHS_G	:	The optional shared secret of the Firmware Group allowing the computation of KS3[9]. Shall be padded with 0 if not present.
$UUID_G$	:	Optional Firmware Group Identifier defined in section 2.3. Shall be padded with 0 when not present.

The Firmware Maker shall generate a new public Firmware Identifier ($UUID_I$) for each Firmware Update. This identifier shall be different from the one used in any previous Firmware Loading / Firmware Update.

For a given $UUID_P$, the IDS shall generate a unique $UUID_I$ for each new Firmware Session.

The following parameters shall not be stored in the Firmware Session:

NUM_SEG	:	The number of segments to load. The minimum value is 0. In that case, no Firmware is transported; only administrative operations are conveyed.
ID_AUTH_G		Boolean true (1) if ISID shall be verified (authentication of IDS_2), else false (0) then ISID is not verified (identification of IDS_2).

The first 16 bits of the OP_{IMA} shall be encoded using Operation Codes listed in Table 7-3. The following 16 bits are the binary complement of the first 16 bits.

Table 7-3: Operation Codes

	Scope			Mask						Bit Rank	
				ARP State			OFL State				
Operation Code Name	**OFL Authority scope**	**TRSH Maker scope**	**Firmware Session scope**	**BLANK**	**LOCKED**	**UNLOCKED**	**TERMINATED**	**DISABLED**	**ENABLED**	**Allowed**	**Complement**
eEraseFirmware	●	●	●	●	●	●		●	●	0	16
eDeleteFirmwareSession	●	●	●	●	●	●		●	●	1	17
eLoadFirmware	●	●	●	●	●	●		●	●	2	18
eDisableFirmware	●	●	●	●	●	●		●	●	3	19
eEnableFirmware	●	●	●	●	●	●		●	●	4	20
eBackupRestoreFirmwareAllowed	●	●	●	●	●	●		●	●	5	21
eTransferRights	●			●	●	●		●	●	6	22
eRFU										7	23
eUnlockOFL	●	●		●	●			●	●	8	24
eDisableOFL	●	●			●			●	●	9	25
eEnableOFL	●	●			●	●		●	●	10	26
eUpdatePolicy	●	●	●		●			●	●	11	27
eTerminateOFL		●			●			●	●	12	28
eUpdateFirmwareAllowed	●	●	●	●	●	●		●	●	13	29
eLocalFirmwareDeleteAllowed	●	●	●	●	●	●		●	●	14	30
eRFU1										15	31

● Indicates that the operation is allowed.

The grayed cells in the Firmware Session scope column define the information to store in the Firmware Session data defined in section 8.5.

The grayed cells in the TRSH Maker scope column define the information to store as OFL data defined in section 8.7.

The following ASN.1 description defines the OFL Operation Codes used for the coding of OP_{IMA}.

```
-- ASN1START
Operation-code                      ::= [PRIVATE 15] BIT STRING
{
eEraseFirmware(0),                  --Erase the Firmware
eDeleteFirmwareSession(1),          --Delete the Firmware Session
eLoadFirmware(2),                   --Load the Firmware
eDisableFirmware(3),                --Disable the Firmware
eEnableFirmware(4),                 --Enable the Firmware
eBackupRestoreFirmwareAllowed(5),   --Allow restore backup
eTransferRights(6),                 --OFL Authority transfert of rights
eRFU(7),                            --RFU
eUnlockOFL(8),                      --No OFL Authority
eDisableOFL(9),                     --Disable OFL
eEnableOFL(10),                     --Enable OFL
eUpdatePolicy(11),                  --Update local policy
eTerminateOFL(12),                  --Terminate OFL
eUpdateFirmwareAllowed(13),         --Firmware Update allowed
eLocalFirmwareDeleteAllowed(14),    --Local Firmware Delete Allowed
eRFU1 (15),                         --RFU
--Complement
eCEraseFirmware(16),                --Erase the Firmware
eCDeleteFirmwareSession(17),        --Delete the Firmware Session
eCLoadFirmware(18),                 --Load the Firmware
eCDisableFirmware(19),              --Disable the Firmware
eCEnableFirmware(20),               --Enable the Firmware
eCBackupRestoreFirmwareAllowed(21), --Allow restore/backup
eCTransferRights(22),               --OFL Authority transfert of rights
eCRFU (23),                         --RFU
eCUnlockOFL(24),                    --No OFL Authority
eCDisableOFL(25),                   --Disable OFL
eCEnableOFL(26),                    --Enable OFL
eCUpdatePolicy(27),                 --Update local policy
eCTerminateOFL(28),                 --Terminate OFL
eCUpdateFirmwareAllowed(29),        --Firmware Update allowed
eCLocalFirmwareDeleteAllowed(30),   --Local Firmware Delete Allowed
eCRFU1(31)                          --RFU
                                    } (SIZE(32))
-- ASN1STOP
```

The following tables define the mask values used in section 9. The following parameters are generated using the algorithm detailed in Annex E.

```
-- ASN1START
eARPINIT-OFLA                       Operation-code::= '11111110111101100000000100001001'B
eARPINIT-TRSH-MAKER                 Operation-code::= '11111100111111100000001100000001'B
eARPINIT-DEFAULT                    Operation-code::= '11111100000101100000001111101001'B
eMASKARP-STATE-BLANK                Operation-code::= '11111110100001110000000101111000'B
eMASKARP-STATE-LOCKED               Operation-code::= '11111110111111100000000100000001'B
eMASKARP-STATE-UNLOCKED             Operation-code::= '11111111001001100000000011011001'B
eMASKOFL-STATE-TERMINATED           Operation-code::= '00000000000000001111111111111111'B
eMASKOFL-STATE-ENABLE               Operation-code::= '11111110111111100000000100000001'B
eMASKOFL-STATE-DISABLE              Operation-code::= '11111110111111100000000100000001'B

ePOLICYUPDATE                       Operation-code::= '00000100000001101111101111111001'B
-- ASN1STOP
```

Table 7-4: Mask Value vs. ARP States

ARP Mask Name	ARP State	$MASK_{ARP\_STATE}$
$MASK_{ARP\_STATE\_BLANK}$	BLANK	eMASKARP-STATE-BLANK
$MASK_{ARP\_STATE\_LOCKED}$	LOCKED	eMASKARP-STATE-LOCKED
$MASK_{ARP\_STATE\_UNLOCKED}$	UNLOCKED	eMASKARP-STATE-UNLOCKED

Table 7-5: Mask Value vs. OFL States

OFL Mask Name	OFL State	$\text{MASK}_{\text{OFL\_STATE}}$
$\text{MASK}_{\text{OFL\_STATE\_TERMINATED}}$	TERMINATED	eMASKOFL-STATE-BLANK
$\text{MASK}_{\text{OFL\_STATE\_DISABLE}}$	DISABLED	eMASKOFL-STATE-ENABLE
$\text{MASK}_{\text{OFL\_STATE\_ENABLE}}$	ENABLED	eMASKOFL-STATE-DISABLE

Table 7-6: ARP_{INIT}

ARP INIT Name	Authority Name	ARP_{INIT}
$\text{ARP}_{\text{INIT\_OFLA}}$	OFL Authority	eARPINIT-OFLA
$\text{ARP}_{\text{INIT\_TRSH\_MAKER}}$	TRSH Maker	eARPINIT-TRSH-MAKER
$\text{ARP}_{\text{INIT\_DEFAULT}}$	Default	eARPINIT-DEFAULT
$\text{ARP}_{\text{LOCAL\_ARP\_TRSHM}}$	Default	eARPLOCAL-ARP-TRSHM

Table 7-7: Local Policy Mask

Name	Value
$\text{POLICY}_{\text{UPDATE}}$	ePOLICYUPDATE

Table 7-3 defines the possible Operation Codes policy. All Operation Codes can be combined (ORing) unless the combined operations lead to an inconsistency (e.g. Load & Delete of session). The Operation Code is a 32-bit word. In order to offer the implementer the opportunity to implement counter-measures against software attacks during the use of Operation Codes and their masks, the first 16 bits allow the operation and the subsequent 16 bits are the complement of the first 16.

7.4.1.2 Segment Descriptor Structure

Table 7-8 illustrates the Segment Descriptor Structure included in an Image that corresponds to the parameter of the OFL_CHANGE_SEGMENT command. There are NUM_SEG occurrences of the Segment Descriptor Structure in the Image Body.

Table 7-8: Segment Descriptor Structure

		Description	Length in Bytes
SDS	K_{SDS} encrypted	Segment operation (1)	4
		Length of the segment L_{SEG} (1)	4
		Cryptographic Algorithm identifier CA_{SEG} (1)	4
		Address Location of the segment A_{SEG} (1)	4
		Segment Integrity check value H_{SEG}	16
		Initialization Vector Integrity check IV_{SEG}	16
		Segment Encrypting Key K_{SEG}	16/32 (2)
		Initialization Vector Encrypting Key EIV_{SEG}	16/32 (2)
Trailer		Value Integrity check H_{SDS}	16
		Segment descriptor rank	4
Total			**100/132(2)**

(1) Endianness according to the endianness of the targeted Secure Hardware Platform.

(2) The size of the key depends on the Cryptographic Algorithm identifier CA_{SEG}.

The key K_{SDS} is computed from the rotation of the keys K_{IMD} as defined in section 4.2.3 with the following rule: Before each SDS encryption, the initial encryption key K_{SDS} is rotated as defined in section 4.2.3; the number of key rotations corresponds to the rank of the SDS. The SDS is encrypted with the key K_{SDS} by using eGCM-AES-128 or eGCM-AES-256 as defined in section 4.2.2. The algorithm used to encrypt the SDS is indicated in the encryption type of the Image Binder Descriptor in section 7.4.2.

The parameters of the SDS are defined in the following table:

Table 7-9: Segment Descriptor Structure Parameters

Parameter	Definition
SO_{SEG}	Type of operations to apply to the Image Segment. '00000001' for Loading '00000002' for Updating
L_{SEG}	Length of the Firmware Segment and the Image Segment; only multiples of 16 are allowed.
A_{SEG}	Address Location of the segment in the memory space; only multiples of 16 are allowed.
H_{SEG}	Integrity check of the corresponding segment.
K_{SEG}	Key used for decrypting the Image Segment.

Parameter	Definition
CA_{SEG}	Cryptographic Algorithm identifier. FFFE 0001 for eGCM-AES-128 FFFD 0002 for eGCM-AES-256
IV_{SEG}	Initialization Vector for Integrity check.
EIV_{SEG}	Initialization Vector for Encrypting key.
H_{SDS}	Segment Descriptor Structure integrity check.

The Image Maker initially generates the SDS, then encrypts it using the transport key KT_{SSD}.

KT_{SSD} is confidentially known by the IDS.

The SDS is decrypted with the key KT_{SSD} within the HSM_IDS$_2$ and then re-encrypted by using the key K_{SDS}, and H_{SDS} is updated accordingly. $H_{SDS,}$ the integrity check of the corresponding SDS, is verified by using eGCM-AES-128 or eGCM-AES-256.

The definition of KT_{SSD} is out of scope of this document and is a convention between the HSM_IDS$_2$ provider and the Image Maker.

7.4.1.3 Image Segment

The Image Segment is a collection of 128-bit blocks encrypted using eGCM-AES-128 or eGCM-AES-256 as defined in section 4.2.2. Encryption is performed by the Image Maker with K_{SEG}, IV_{SEG}, and EIV_{SEG} as defined in section 7.4.1.2, and the integrity check of the Image Segment is provided in H_{SEG}. The algorithm used to encrypt the Segment is indicated in the SDS in CA_{SEG}, as defined in section 7.4.1.2. The length of the Image Segment corresponds to the length of the segment to load. The structure is the content of the OFL_LOAD_SEGMENT parameter command. There are NUM_SEG Image Segments in the Image Body.

7.4.2 Image Binder Descriptor

The IMAGE-BINDER-DESCRIPTOR is part of the OFL_DO_OPERATE parameter defined in section 6.3.1.

The following ASN.1 description defines the Binder Descriptor.

```
-- ASN1START
UUIDI                                      ::= UUID
                                           -- Public Firmware Identifier (UUIDI)
Encryption-Type                            ::= [PRIVATE 16]ENUMERATED
{
eGCM-AES-128                               (0),
                                           --Encryption type eGCM-AES-128

eGCM-AES-256                               (1)
                                           --Encryption type eGCM-AES-256

}

IMAGE-BINDER-DESCRIPTOR ::= [APPLICATION 3] SEQUENCE
{
aImageType                                 CHOICE
{
   aUnicast                                [0]SEQUENCE
      {
      aFirmwareId                          UUIDI,
      aIDS2-Token                          Token,
      aIDS2-CertificationPath              [0]CertificationPath,
      aM-Time-Stamp                        EncryptedBlock OPTIONAL
                                           --Time Stamp
      },
   aMulticast                          [1] SEQUENCE
      {
      aGroupId                         UUIDG,
      aAuthRequired                    SEQUENCE
           {
           aIDS2-Token                 Token,
           aIDS2-CertificationPath     [0]CertificationPath,
           aM-Time-Stamp               EncryptedBlock
                                       --Time Stamp
           } OPTIONAL
      }
},
aEncryption-Type                       Encryption-Type,
                                       --Type of Encryption
aM-ARP                                 EncryptedBlock OPTIONAL,
                                       --encrypted CERT.ARP.ECDSA
aM-IMD                                 EncryptedBlock
                                       --Image Descriptor IMD
}
-- ASN1STOP
```

Where:

aImageType	:	Type of the Image which shall be either a unicast Image (aUnicast) or a multicast Image (aMulticast)
aFirmwareId	:	The Firmware Identifier(UUID$_I$)
aGroupId	:	The Firmware Group Identifier (UUID$_G$)
aIDS$_2$-Token	:	An ATK.IDS$_2$.ECKA Authentication Token; a Token as defined in section 6.2.1 and in section 11.3.3.1 that contains in its aATK-Content an ATK-IDS$_2$-ECKA
aIDS$_2$-CertificationPath	:	List of the Certificates of the IDS$_2$ Certification Path originating from PK.CI$_{OFL}$.ECDSA public key and ending at the IDS$_2$ Certificate (CERT.IDS$_2$.ECDSA)
aEncryption-Type	:	Type of encryption used to encrypt the aM-IMD, aM-ARP, aM-Time-Stamp, and the SDS
aM$_{IMD}$	:	Contains the cryptogram and the integrity check of the IMD by using the key K$_{IMD}$
aM$_{TIME-STAMP}$	:	Contains the cryptogram and the integrity check of the Time-stamp as defined in section 11.3.4 by using the key K$_{TIME-STAMP}$
aM$_{ARP}$	:	Contains the cryptogram and the integrity check of the CERT.ARP.ECDSA Certificate by using the key K$_{ARP}$

8 Firmware Management

8.1 Requirements

8.1.1 Firmware Loading Requirements

The Secure Hardware Platform shall only execute a Firmware loaded by the OFL.

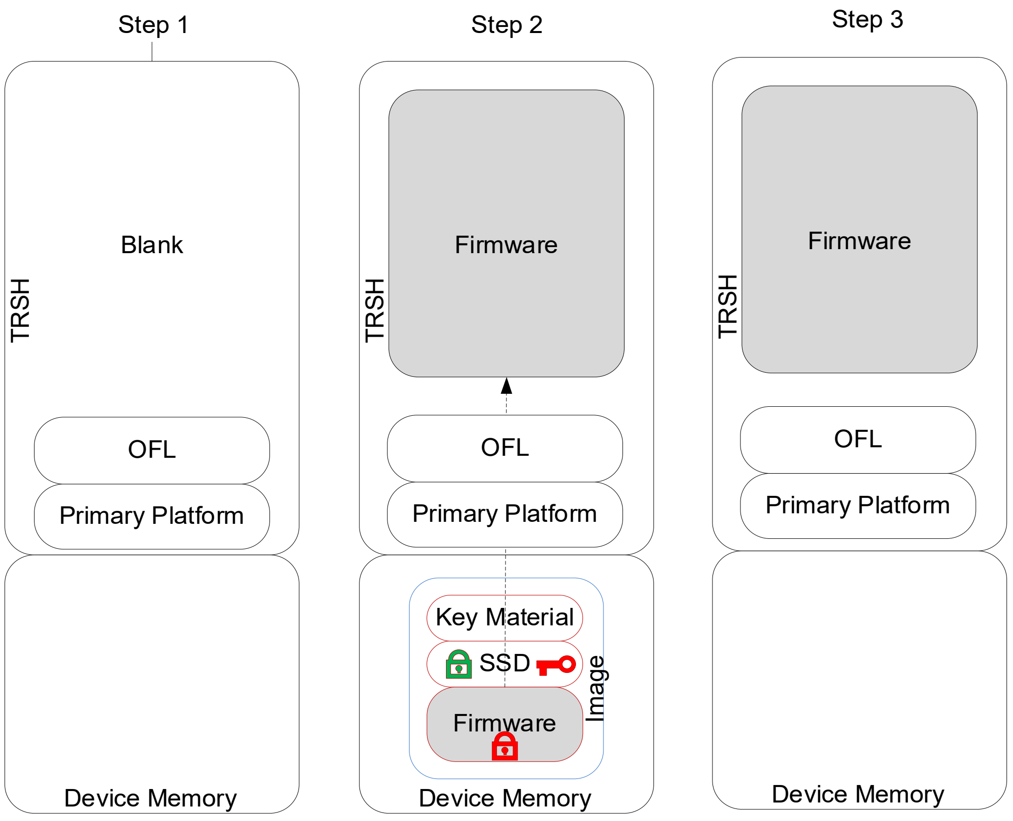
Only one Firmware stored outside the TRSH shall be loaded at a time in the TRSH.

There shall be a single Firmware Loading per Firmware.

Note: The case where a TRSH has multiple Firmware in the TRSH is out of scope of the specification.

Before loading a new Firmware in the TRSH, the existing Firmware, if any, shall first be deleted or backed up.

Figure 8-1: Firmware Loading Steps



The Firmware Loading procedure is performed in three steps:

1. The TRSH is blank or erased.
2. The Image is loaded from the OFL Agent into the OFL. The OFL Agent executes the commands with parameters extracted from the Image as described in section 7.4 and according to the instructions extracted from the Image Manifest. Finally, the OFL installs the Firmware in the TRSH.
3. After a successful Firmware Loading, a Firmware Session is created. The Image and its Manifest in the device memory are no longer valid and then may be discarded.

8.1.2 Firmware Update Requirements

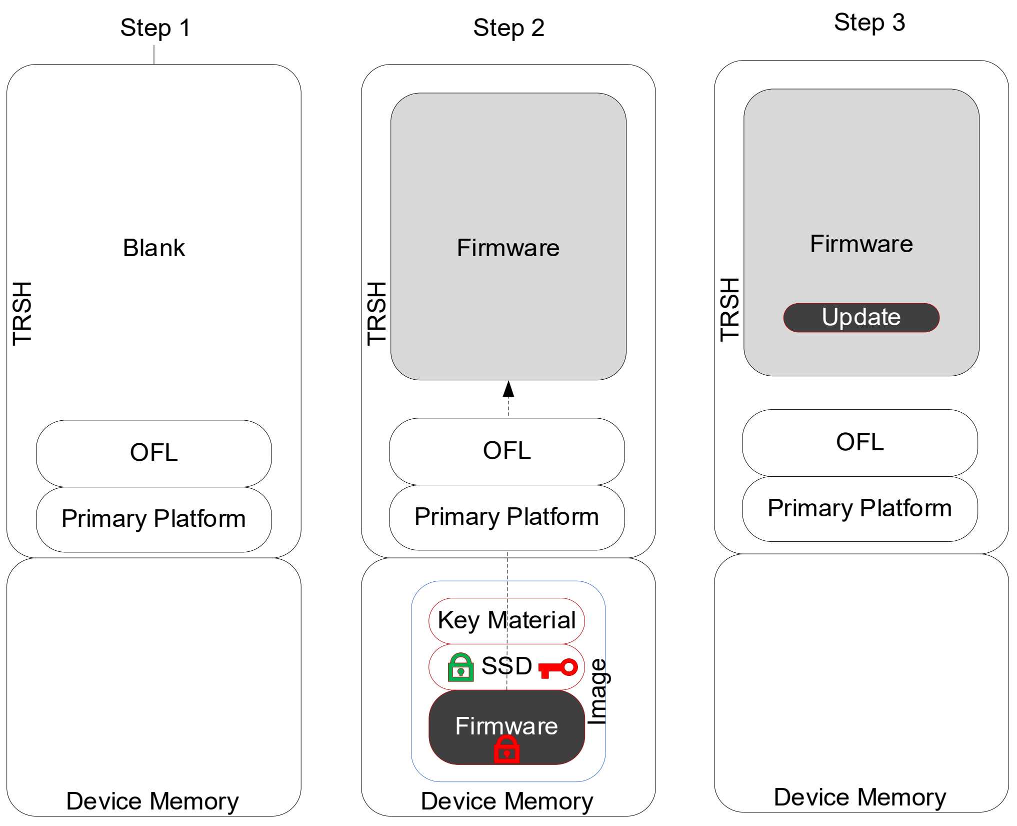
The Secure Hardware Platform shall only execute a Firmware loaded by the OFL. The Firmware Update definition applies as soon as a Firmware Session exists.

Only one Firmware stored outside the TRSH shall be loaded at a time in the TRSH.

There shall be zero or more Firmware Updates for loaded Firmware using the Firmware Session identified by the private Firmware Identifier.

Note: The case where a TRSH has multiple Firmware in the TRSH is out of scope of the specification.

Figure 8-2: Firmware Update Steps



The Firmware Update procedure is performed in three steps:

1. The TRSH is blank or erased.
2. The Firmware is restored in the TRSH.
3. The Image is loaded from the OFL Agent into the OFL. The OFL Agent executes the commands with parameters extracted from the Image as described in section 7.4 and according to the instructions extracted from the Image Manifest. Finally, the OFL updates the Firmware in the TRSH. The Image and its Manifest in the device memory are no longer valid and then may be deleted.

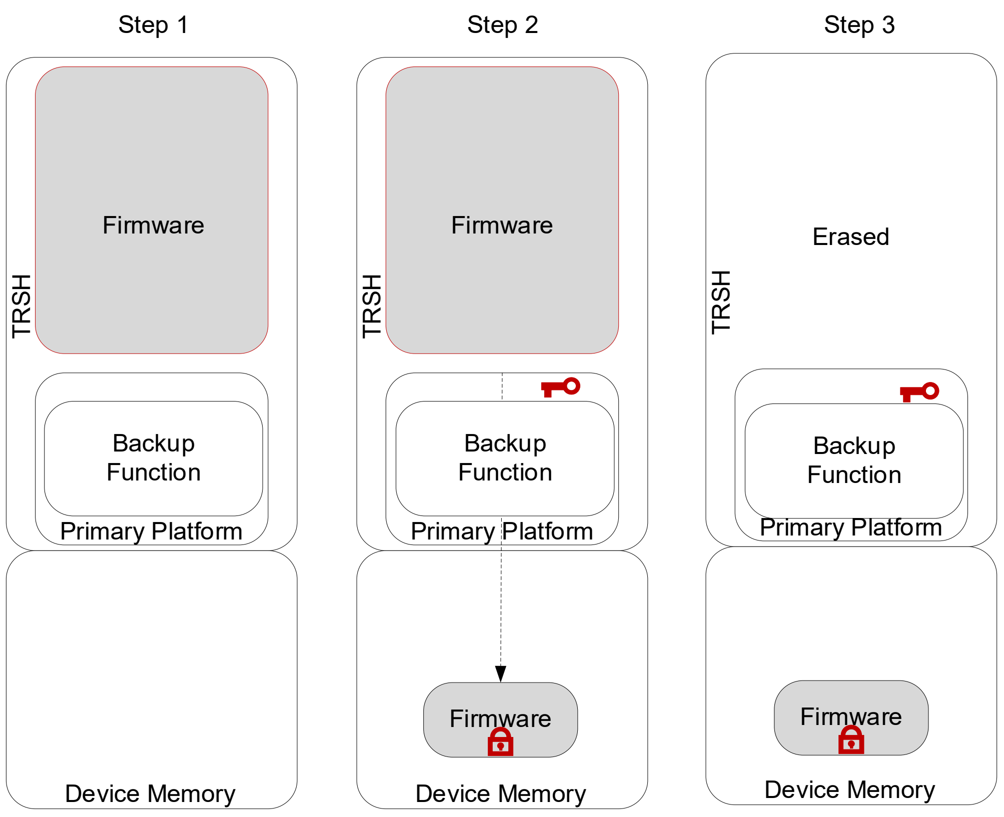
8.1.3 Firmware Backup Requirements

The Firmware, backed up outside the TRSH perimeter, shall be protected in confidentiality, integrity and anti-replay by using means only known by the Secure Hardware Platform.

The means for protecting the Firmware outside the TRSH perimeter shall be diversified by Firmware.

Backup function is optional and may be proprietary.

Figure 8-3: Firmware Backup Steps



The TRSH hosts a Firmware.

The Firmware backup procedure is performed in two steps:

1. Subsequent to a request from the Device Host, a backup function encrypts, saves, and stores the encrypted Firmware into the device memory by using ephemeral keys securely stored in the TRSH. The type and nature of the device memory is use case and implementation dependent.
2. The Firmware in the TRSH is erased.

8.1.4 Firmware Restore Requirement

The encryption mechanism used for the Firmware restore is left to the implementer.

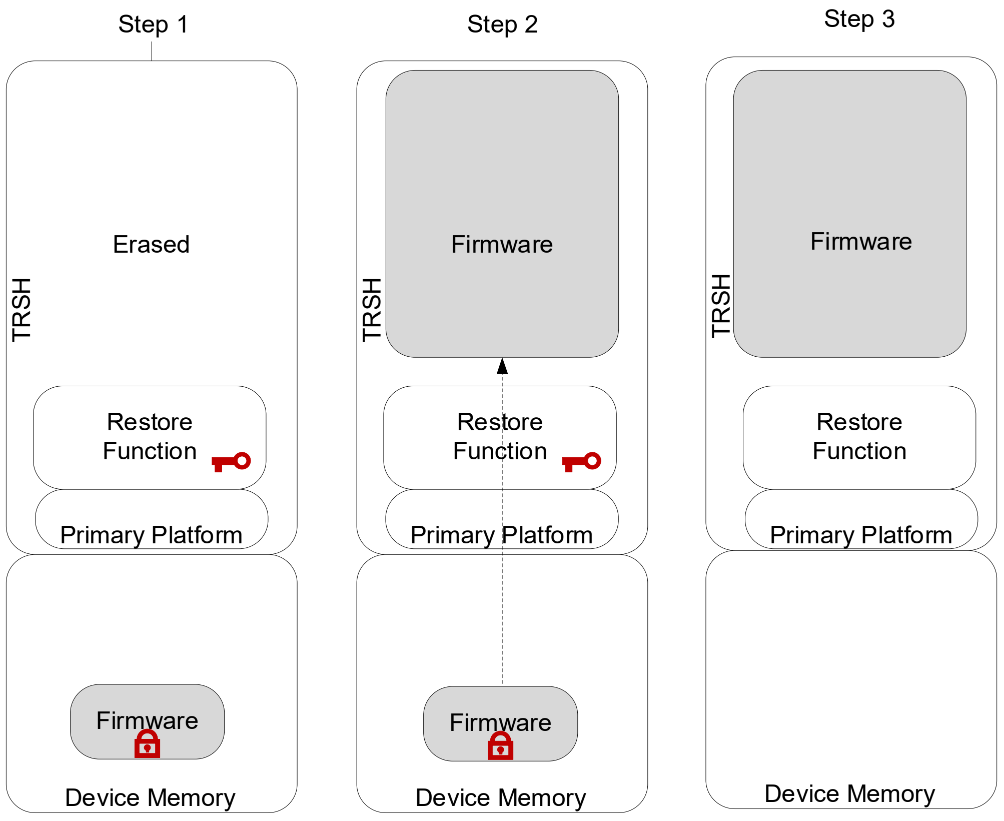
The Secure Hardware Platform shall not execute any Firmware.

The Firmware restored from a storage outside the TRSH perimeter shall be verified in integrity and anti-replay by using means only known by the Secure Hardware Platform.

Before restoring a Firmware in the TRSH, the existing Firmware, if any, shall first be deleted or backed up.

The restore function is optional and may be proprietary.

Figure 8-4: Firmware Restore Steps



The TRSH does not host any Firmware.

The restore procedure is performed in two steps:

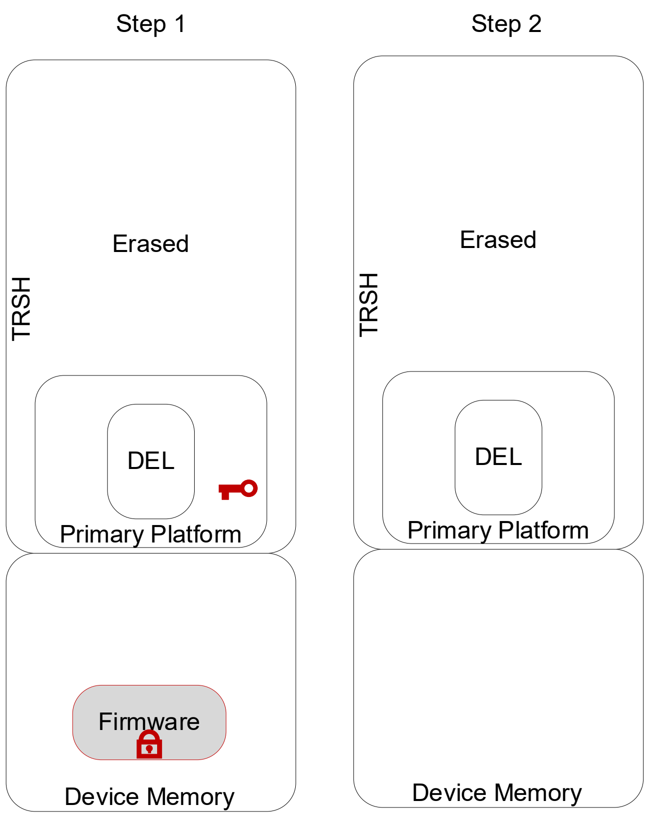
1. Before the restoring of a Firmware the TRSH memory is erased.
2. Subsequent to a request from the Device Host, the Restore function restores the encrypted Firmware from the Device memory by using an ephemeral key hosted in the TRSH memory.
3. The TRSH hosts a Firmware. The TRSH may erase the encrypted Firmware from the device memory and shall destroy the ephemeral key.

8.1.5 Firmware Deletion Requirements

The Secure Hardware Platform shall only execute Firmware loaded by the OFL.

Before deletion of Firmware in the TRSH, the Firmware shall be in the DISABLED state.

Figure 8-5: Firmware Deletion Steps



The Firmware deletion procedure is performed in two steps:

1. Subsequent to a request from the Device Host, the OFL discards the associated Firmware Session and all internal data related to the Firmware.
2. The encrypted Firmware is deleted from the TRSH memory and from the Device memory.

For non-removable TRSH, the implementation of the delete function is mandatory.

8.1.6 Firmware Enabling Requirements

The Image Owner shall be able to set its Firmware to the ENABLED state as defined in section 8.4.3.

8.1.7 Firmware Disabling Requirements

The Image Owner shall be able to set its Firmware to the DISABLED state as defined in section 8.4.3.

8.1.8 Interface Requirements

There shall be a protocol between the OFL and the OFL Agent.

It should support the following interface over the above protocol:

- OFL_DELETE_FIRMWARE: This interface allows the local deletion of a Firmware.

It may support the following interfaces over the above protocol:

- OFL_ENABLE_FIRMWARE / OFL_DISABLE_FIRMWARE: These interfaces allow the local setting of the Firmware to the ENABLED state and DISABLED state, respectively.

8.2 Firmware Family Identifier Management

The restrictions defined in the certificates shall apply as follows:

- If the Firmware Family Identifier of the IMD is not in the list of Firmware Family Identifiers in the certificate signing CERT.IDS$_2$.ECDSA Certificate extension defined in section 11.2.2.2, then the Image Loading shall abort.
- If the Part Number Identifier in the certificate signing CERT.IDS$_2$.ECDSA Certificate extension defined in section 11.2.2.2 is present and if it does not match the Part Number Identifier of the TRSH, then the Image Loading shall abort.

8.3 Access Rights Pattern

8.3.1 Overview

The security protocol defined in section 5 includes the Access Right Pattern (ARP), which allows the OFL Authority or a TRSH Maker to define the OFL Operations that an Image Owner is allowed to request and execute on the OFL.

The OFL may have no OFL Authority if the OFL ARP state is UNLOCKED.

The ARP Certificate issued by the TRSH Maker allows to define operations related to the Firmware Session targeting the Secure Hardware Platform Firmware and the OFL Firmware.

The TRSH Maker ARP Certificate does not impact the ARP and OFL state.

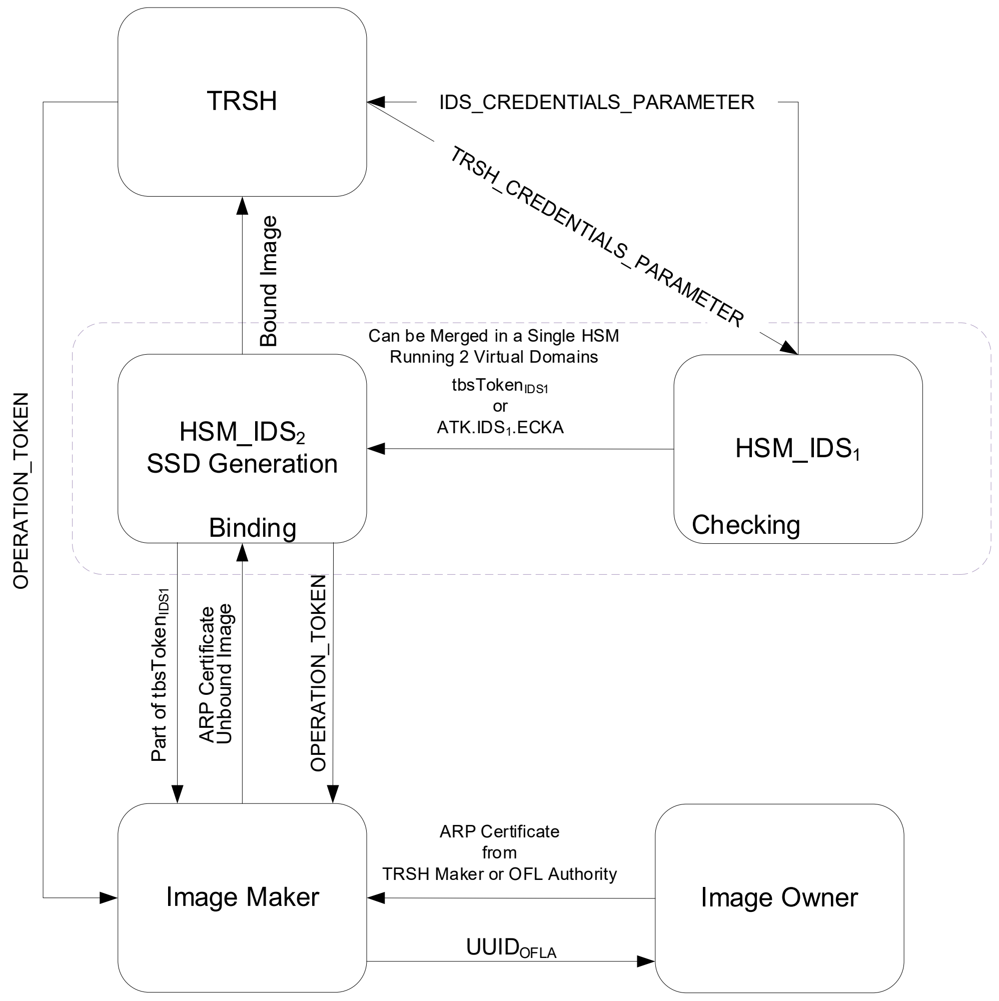
If an ARP Certificate is included in the Image, then that certificate shall be issued by the OFL Authority or the TRSH Maker and shall be verified according to the credentials of their issuers (PK.IMO$_{OFLA}$.ECDSA or PK.TRSH$_M$.ECDSA).

If an ARP Certificate is included in the Image and fails the ARP Certificate verification, then the Image Loading fails, however the Image Session remains.

Only OFL Operations within the Firmware Session scope, as defined in Table 7-3, are allowed as Operation Codes.

Figure 8-6 describes the global flow of the ARP mechanism.

Figure 8-6: Access Rights Pattern Flow


TRSH
IDS_CREDENTIALS_PARAMETER
TRSH_CREDENTIALS_PARAMETER
Bound Image
Can be Merged in a Single HSM Running 2 Virtual Domains
tbsToken$_{IDS1}$ or ATK.IDS$_1$.ECKA
HSM_IDS$_2$ SSD Generation
Binding
HSM_IDS$_1$
Checking
OPERATION_TOKEN
Part of tbsToken$_{IDS1}$
ARP Certificate Unbound Image
OPERATION_TOKEN
ARP Certificate from TRSH Maker or OFL Authority
Image Maker
Image Owner
UUID$_{OFLA}$

The ARP state and the ARP format are defined respectively in sections 8.4.1 and 8.3.2. The procedures to verify the ARP are defined in sections 9.3 and 9.4.

8.3.2 ARP Format

The format of the ARP is defined in section 11.2.12.

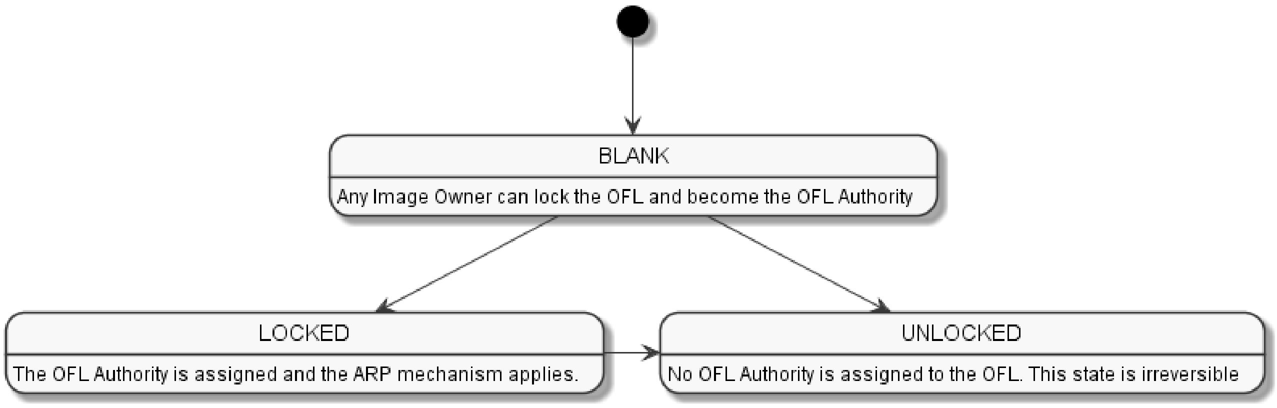
8.4 States

8.4.1 ARP States

The OFL defines three ARP states:

- **BLANK**: Any Image Owner can lock the OFL and become the OFL Authority.
- **LOCKED**: The OFL Authority is assigned and the ARP mechanism applies.
- **UNLOCKED**: No OFL Authority is assigned to the OFL. This state is immutable.

Figure 8-7: OFL ARP States


BLANK
Any Image Owner can lock the OFL and become the OFL Authority
LOCKED
The OFL Authority is assigned and the ARP mechanism applies.
UNLOCKED
No OFL Authority is assigned to the OFL. This state is irreversible

The OFL ARP states are indicated by the values of $UUID_{OFLA}$ as defined in section 2.3:

- $UUID_{OFLA}$ = $[FF]_{16}$ indicates that the OFL ARP state is BLANK.
- $UUID_{OFLA}$ = $[00]_{16}$ indicates that the OFL ARP state is UNLOCKED.

Any other value of $UUID_{OFLA}$ corresponds to the OFL ARP state LOCKED. In this case, $UUID_{OFLA}$ is the UUID of the OFL Authority.

Table 8-1 defines which operations are allowed depending on the OFL ARP state. XXXX and YYYY denote a particular $UUID_{IMO}$ taken into account during a DO_OPERATE Operation. They denote, as well, a particular $UUID_{OFLA}$ before and after a successful DO_OPERATE Operation. ANY denotes any $UUID_{IMO}$.

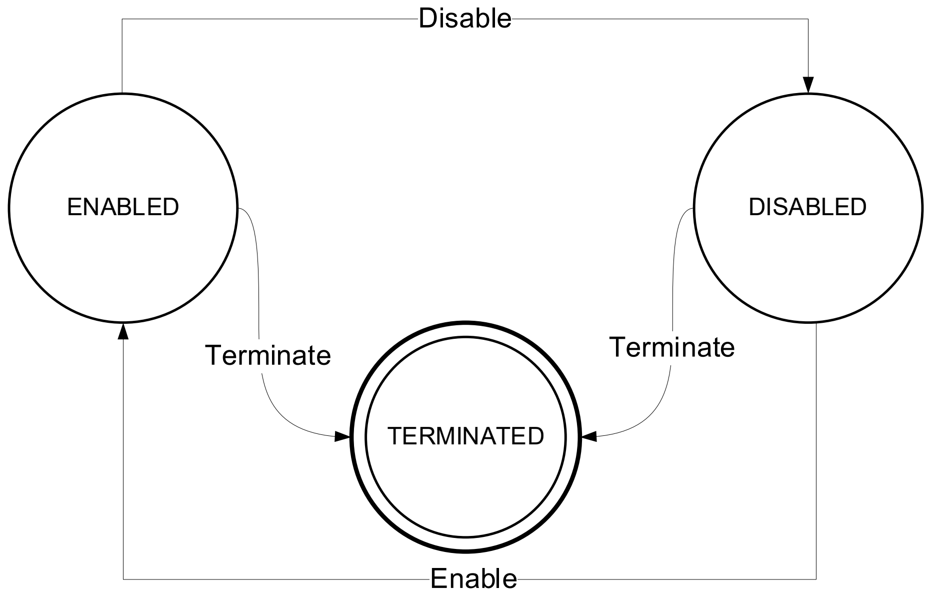
Table 8-1: Operations Rules Depending on OFL ARP State

OFL ARP State	$UUID_{OFLA}$ (OFL Authority)	$UUID_{IMO}$	Operation (DO_OPERATE)	Updated $UUID_{OFLA}$ (after operation)
BLANK	$[FF]_{16}$	ANY	All operations are allowed. Unlock OFL & Lock OFL Operations are described in the rows below.	$[FF]_{16}$
		ANY	Unlock OFL is allowed.	$[00]_{16}$
		XXXX	Lock OFL is allowed.	XXXX
UNLOCKED	$[00]_{16}$	ANY	All operations are allowed except Lock OFL.	$[00]_{16}$
LOCKED	XXXX	ANY	Without an ARP Certificate signed by the OFL Authority assigned to the IMO, operation is rejected.	XXXX
		ANY	With an ARP Certificate signed by the OFL Authority assigned to the IMO, any operation may be allowed. Transfer Rights and Unlock OFL Operations are described in the rows below.	XXXX
		YYYY	Transfer Rights with an ARP Certificate signed by the OFL Authority assigned to the IMO_{YYYY} is allowed.	YYYY
		ANY	Unlock OFL with an ARP Certificate signed by the OFL Authority is allowed.	$[00]_{16}$

8.4.2 OFL States

OFL state is a property of the OFL and is illustrated in Figure 8-8. The OFL shall manage the persistent state machine of the OFL independently of other Firmware Sessions.

Figure 8-8: OFL States



The OFL operates with the following rules:

- When the OFL is executed, the OFL runs.
- This OFL state is persistent across successive power ON/OFF cycles of the TRSH.
- Any access to the whole OFL registry is allowed unless stated.

OFL states are as follows:

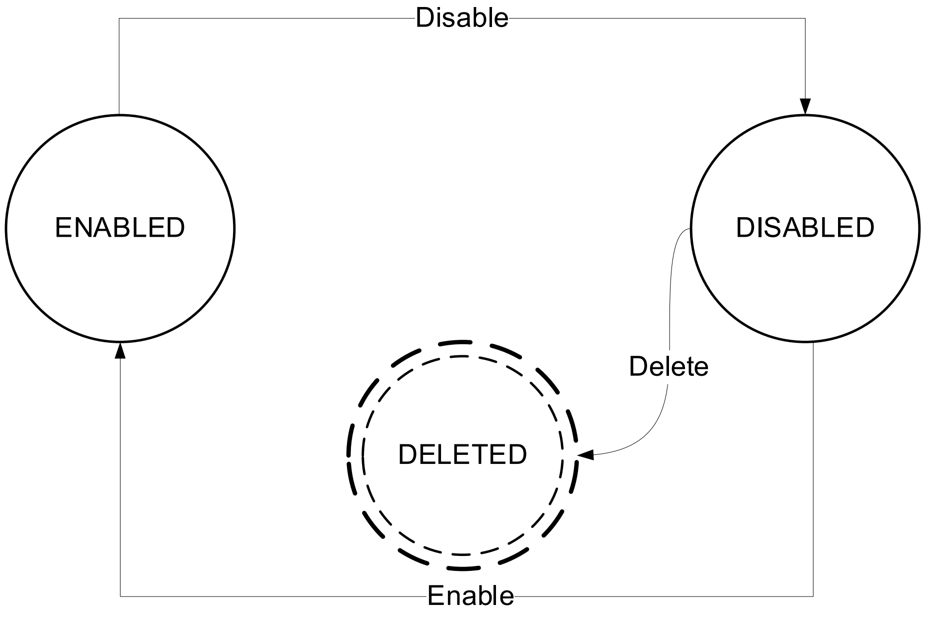
- **ENABLED:** Any Firmware in the ENABLED state may be executed.
- **DISABLED:** When DISABLED, Firmware other than the OFL cannot be executed.
- **TERMINATED:** OFL can no longer perform any operation except access to the OFL registry. This status is irreversible. In the TERMINATED state, the Firmware Session related to OFL and the OFL credentials remains useable; all other Firmware Sessions are no longer useable. All OFL registry accesses are allowed (except for registries that refer to the credentials TRSH_CREDENTIALS_PARAMETER and IDS_CREDENTIALS_PARAMETER). This state is persistent across successive power cycles. In this state, no Firmware can be executed except the OFL itself.

Note: OFL itself refers to all Firmwares needed to start the OFL; e.g. in the case of the VPP ecosystem, this includes low level operating system and related libraries, packages, etc.

8.4.3 Firmware States

Firmware state is a property of a Firmware and is illustrated in Figure 8-9. The OFL shall manage the persistent Firmware state in each Firmware Session.

Figure 8-9: Firmware States



The Firmware states are persistent across successive power cycles.

Firmware states are as follows:

- **ENABLED:** When ENABLED, the Firmware may be executed.
- **DISABLED:** When DISABLED, the Firmware shall not be executed.
- **DELETED:** The transition to DELETED is irreversible. This state is abstract because it results in the deletion of the Firmware Session and the Firmware itself.

8.4.4 Firmware Execution

If the Firmware state and OFL state are both ENABLED, then the Firmware may be executed and may run.

The means and rules for running and stopping a Firmware are out of scope of this specification and defined in the specification of the concrete platform. When stopped, the running Firmware shall be reliably removed from the TRSH execution memory by means that are system and/or implementation dependent and out of scope of this specification.

Note: As an example, in *GlobalPlatform Virtual Primary Platform – Concepts and Interfaces* ([VCI]), only one Firmware is allowed to be executed at a given time. The running Firmware is called VPP Application

8.5 Firmware Session Data

This section describes the non-volatile storages containing the data of the Firmware Sessions.

The Firmware Session data dedicated to a given loaded Firmware contains:

- $UUID_P$: The private Identifier of the Firmware in the IMD in section 7.4.
- $UUID_I$: The public Identifier of the Firmware in the IMD in section 7.4.
- $UUID_G$: The Firmware Group Identifier (optional) as defined in section 2.3
- $UUID_F$: The Firmware Family Identifier as defined in section 2.3.
- $UUID_M$: The Identifier of the Firmware Maker as defined in section 2.3.
- VER_CURR: The current Firmware version as defined in section 7.4.
- Firmware State: The state of the Firmware (ENABLED or DISABLED) as defined in section 8.4.3.
- $LOCAL\_ARP_N$: The operations allowed to the OFL on the Firmware:
  - Firmware Update allowed.
  - Local Delete of the Firmware allowed.
  - Backup/restore of the Firmware allowed.
- SHS_G: The shared secret used for the KS3[9] Firmware Group key derivation (optional).
- ID_AUTH_G: Indicates if ISID shall be verified (authentication of IDS_2) (optional).

Counter: A 32-bit counter which is set to 0 at the creation of the Firmware Session and shall be incremented after each change of the Firmware state due to an OFL Operation.

Multiple Firmware Sessions can coexist simultaneously (the maximum number of Firmware Sessions is implementation dependent). Each of those Firmware Sessions is identified by the public Firmware Identifier assigned at the Firmware Loading.

8.6 OFL Operations

Table 8-2 describes the OFL Operations.

Table 8-2: OFL Operation Descriptions

Operation	Description
Firmware Update Allowed	This operation allows the Firmware Update or Secure Hardware Platform Firmware Update.
Local Firmware Delete Allowed	Allows local delete (e.g. the end user may request local delete through the OFL Agent).
Delete Firmware Session	Deletes the Firmware Session and the firmware itself from the TRSH and optionally from the device if backup is supported. $UUID_P$ shall correspond to the $UUID_P$ in the Firmware Session to be deleted.
Disable Firmware	Sets the Firmware to the DISABLED state (see section 8.4.3). The Firmware cannot be executed. $UUID_P$ shall correspond to the $UUID_P$ of the Firmware Session to allow the operation on the corresponding Firmware.
Disable OFL	Allows an Image Owner to switch the OFL to the DISABLED state (see section 8.4.1).
Enable Firmware	Sets the Firmware to the ENABLED state (see section 8.4.3). $UUID_P$ shall be equal to the one stored in the Firmware Session to allow the operation on the corresponding Firmware.
Enable OFL	Allows an Image Owner to switch the OFL to the ENABLED state (see section 8.4.1).
Erase Firmware	Erases a firmware in the TRSH only. $UUID_P$ shall correspond to the $UUID_P$ in the Firmware Session of the Firmware to be erased. The firmware could be loaded again locally if a backup has been made.
Load Firmware	Enables the Firmware Loading into the TRSH memory.
Lock OFL / Transfer Rights	Transfers the OFL Authority role to another Image Owner and set the ARP to the LOCKED state (see section 8.4.1).
Terminate OFL	Sets the OFL to the TERMINATED state (see section 8.4.1). This operation is irreversible.
Unlock OFL	Sets the ARP to the UNLOCKED state (see section 8.4.1). This operation is irreversible.
Update Local Policy	Sets or updates the list of operations allowed by the Local Policy (see section 9.4). Note that it always restricts the authorized Firmware Management operations.

8.7 OFL Data

This section describes the non-volatile storages containing the data for the management of the OFL state.

The OFL management storage contains in addition to the long terms keys defined in section 5.1.1:

- The public key and the OFL Authority Identifier, respectively PK.IMO$_{OFLA}$.ECDSA and UUID$_{OFLA}$.
- The current Local ARP as defined in section 8.3
- The OFL ARP state defined in section 8.4.1
- The OFL state defined in section 8.4.2 per CI$_{OFL}$ the associated set of:
  - The long terms static keys listed in section 5.1.1
  - The PN (CERT.PN.ECDSA) and FAC (CERT.FAC.ECDSA) Certificates

9 Procedures

9.1 Procedure to Bind an Image to the OFL

- The TRSH provides its credentials to the HSM_IDS_1 operating within the IDS (Image Delivery Server).
- The HSM_IDS_1 verifies the TRSH Credentials using ATK.OFL.ECKA and that CERT.OFL.ECDSA is in the CI_{OFL} Certificates chain. CODE_M may be a genuine Secure Hardware Platform Pseudonym Identifier as defined in section 10.2.
- If the TRSH Credentials verification is successful, then HSM_IDS_1 shall generate and provide the $ATK.IDS_1.ECKA$ Authentication Token or $tbsToken_{IDS1}$ to HSM_IDS_2. $ATK.IDS_1.ECKA$ and $tbsToken_{IDS1}$ are defined in section 11.3.3.2. The following rules shall apply:
    - If HSM_IDS_1 and HSM_IDS_2 are not confined in a security perimeter managed by the same entity, then HSM_IDS_1 shall provide an $ATK.IDS_1.ECKA$ Authentication Token authenticating the data extracted from the ATK.OFL.ECKA Token. The verification of this signature by HSM_IDS_2 is mandatory.
    - If HSM_IDS_1 and HSM_IDS_2 are confined in a security perimeter managed by the same entity, then HSM_IDS_1 may provide a $tbsToken_{IDS1}$.
    - The level of security between HSM_IDS_1 and HSM_IDS_2 is implementation dependent and the communication between both shall ensure the confidentiality and the integrity of the $ATK.IDS_1.ECKA$ Authentication Token or $tbsToken_{IDS1}$ transfers.
- The Image Maker provides to the HSM_IDS_2 the Image Body as defined in section 7.4.1. This Image Body has its SSD encrypted with the transport key KT_{SSD}. The means for retrieving the aforesaid KT_{SSD} is dependent of the interface between the Image Maker and the IDS_2 and is out of scope of this specification.
- If the ARP Certificate is required, then it is retrieved from the OFL Authority granting the rights to operate the OFL on the TRSH. The ARP Certificate is defined in section 11.2.12 and shall be provisioned into HSM_IDS_2.
- HSM_IDS_2 performs the following operations:
    - If provided, verify $ATK.IDS_1.ECKA$ with $PK.IDS_1.ECDSA$.
    - Generate the ephemeral key $KS2^9$ necessary to decrypt the OFL_CHANGE_SEGMENT parameters.
    - Decrypt the Image SSD from the Image Maker with the key KT_{SSD} to retrieve the IMD and the SDS. Retrieve the keys K_{SDS} and K_{IMD} as the transport keys from the IMD. The decryption of the SSD is performed as follows:
        - The key K_{SDS} is computed from the key rotation (as defined in section 4.2.3) of the key K_{IMD}.
        - Before each SDS decryption, the initial encryption key K_{SDS} is rotated as defined in section 4.2.3; the number of key rotations corresponds to the rank of the SDS. The determination of the key length between the IDS and the Image Maker is out of this specification.
        - The SDS is decrypted with the key K_{SDS} by using the decryption function as defined in section 4.2.2.
    - Re-encrypt the SSD by using the keys K_{SDS} and K_{IMD} deduced from the key matrix $KS2^9$ as defined in Table 5-3, Ephemeral Keys.

- Generate the OFL_DO_OPERATE parameter as defined in section 6.3.1 with the updated Image Body.
- OFL shall compute two Operation Tokens (ATK_OFL_DO_OPERATE and ATK_OFL_DELETE_SESSION) proving the completion of an OFL Operation as defined in section 5.4.5.

ATK_OFL_DO_OPERATE may be read by the OFL Agent from the OFL registry OPERATION_TOKEN when the associated initial Image Loading has been successfully completed (see section 10.3).

ATK_OFL_DELETE_SESSION may be read by the OFL Agent from the OFL registry OPERATION_TOKEN when the associated Firmware Session deletion has been successfully completed (see section 10.3).

If the OFL ARP state is LOCKED, then the Image Binding requires an ARP Certificate (defined in section 11.2.12). The ARP Certificate shall be provided by the OFL Authority dully identified by the $UUID_{OFLA}$ identifier.

The OFL Authority is an Image Owner that has transferred its public key PK.IMO.ECDSA and its $UUID_{IMO}$ into the OFL persistent memory as $PK.IMO_{OFLA}$.ECDSA and the $UUID_{OFLA}$.

9.2 Procedure for the Multicast Firmware Update

9.2.1 Description

The multicast Firmware Update allows the update of Firmware sharing the same Group Identifier within multiple TRSH with the same Firmware.

In order to install the needed parameters to allow consecutive multicast Firmware Updates, an initial Firmware Loading shall be successfully executed. A Firmware Session is created after this Firmware Loading, as in the Image Binder Descriptor the MULTICAST Firmware is indicated.

The parameters are installed using the registration operation below. The parameters are either contained in the initial Firmware or in a subsequent Firmware that will perform an update (in unicast mode) using the same Firmware Session.

9.2.2 Registration of the Multicast Firmware Update Parameters

SHS_G and ID_AUTH_G shall be stored in the Firmware Session if the Image Loading is successful.

SHS_G is the shared secret used as a seed for computing $KS3^9$ as the Firmware Group key for multicast Firmware Update.

Only if ID_AUTH_G is present then ISID shall be verified (authentication of the IDS).

9.2.3 Multicast Firmware Update

The multicast Firmware Update operation requires the following steps:

- $KS3^9$ key matrix is derived from SHS_G.
- $KS2^9 = KS3^9$.

In identification mode (if the ID_AUTH_G is present), $M_{TIME\text{-}STAMP}$ is not decrypted and the Certificate time validity shall not be checked.

For a given Firmware Session related to $UUID_G$, checks of the $UUID_I$ and the $UUID_P$ are not performed.

9.3 Procedure to Verify the ARP

The OFL shall accept an operation from any Image provided by an authenticated IDS in any of the following cases:

- If the OFL ARP state is BLANK or UNLOCKED and the operation is not restricted to the TRSH Maker
- If the OFL ARP state is LOCKED and the operation does not required the verification of the ARP Certificate
- If the OFL ARP state is LOCKED and the operation is authorized in the ARP Certificate, which shall be verified by using PK.IMO$_{\text{OFLA}}$.ECDSA

CERT.ARP.ECDSA shall be verified before any OFL Operation intended to:

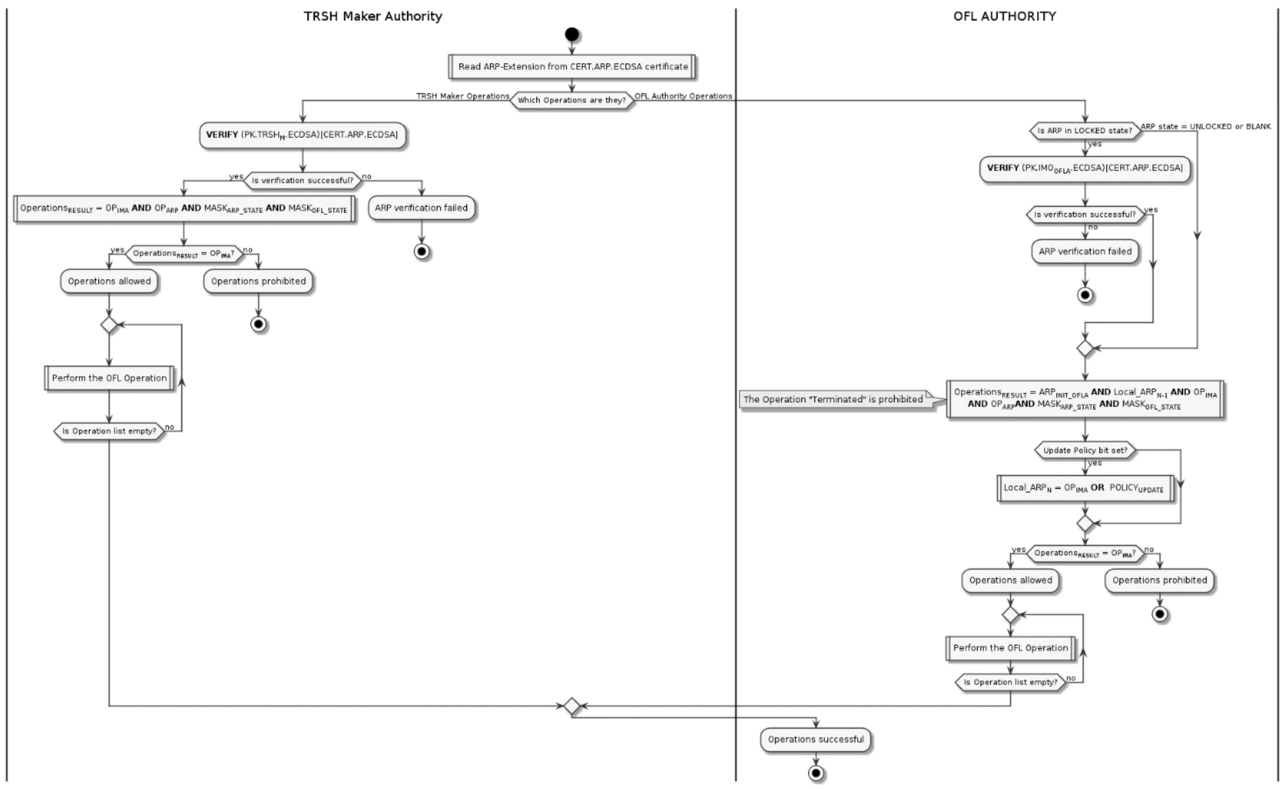
- Modify the OFL state
- Modify the OFL Authority

The following data is involved in ARP verifications:

- LOCAL_ARP$_{\text{N-1}}$: The current (prior to the operation) local ARP in the OFL Data
- OP$_{\text{IMA}}$: The requested operation in the Image Loading defined in section 7.4.1.1
- OP$_{\text{ARP}}$: Operation allowed provided in the ARP certificates as defined in section 11.2.12
- MASK$_{\text{ARP\_STATE}}$: ARP Mask depending of the current ARP state defined in Table 7-4
- MASK$_{\text{OFL\_STATE}}$: ARP Mask depending of the current OFL state defined in Table 7-5
- ARP$_{\text{INIT}}$: ARP Mask depending of the Authority involved defined in Table 7-6
- Update Policy bit: Operation Code bit for Update Policy
- Operation$_{\text{RESULT}}$: The resulting set of operations allowed for the Image Loading

Figure 9-1 illustrates the processing of the CERT.ARP.ECDSA Certificate.

Figure 9-1: ARP Certificate Processing


TRSH Maker Authority
OFL AUTHORITY
Read ARP-Extension from CERT.ARP.ECDSA certificate
TRSH Maker Operations
Which Operations are they?
OFL Authority Operations
VERIFY (PK.TRSH$_M$.ECDSA)[CERT.ARP.ECDSA]
yes
Is verification successful?
no
Operations$_{RESULT}$ = OP$_{IMA}$ **AND** OP$_{ARP}$ **AND** MASK$_{ARP\_STATE}$ **AND** MASK$_{OFL\_STATE}$
ARP verification failed
yes
Operations$_{RESULT}$ = OP$_{IMA}$?
no
Operations allowed
Operations prohibited
Perform the OFL Operation
Is Operation list empty?
no
Is ARP in LOCKED state?
ARP state = UNLOCKED or BLANK
yes
VERIFY (PK.IMO$_{OFLA}$.ECDSA)[CERT.ARP.ECDSA]
Is verification successful?
yes
no
ARP verification failed
The Operation "Terminated" is prohibited
Operations$_{RESULT}$ = ARP$_{INIT\_OFLA}$ **AND** Local_ARP$_{N-1}$ **AND** OP$_{IMA}$ **AND** OP$_{ARP}$**AND** MASK$_{ARP\_STATE}$ **AND** MASK$_{OFL\_STATE}$
Update Policy bit set?
yes
Local_ARP$_N$ = OP$_{IMA}$ **OR** POLICY$_{UPDATE}$
yes
Operations$_{RESULT}$ = OP$_{IMA}$?
no
Operations allowed
Operations prohibited
Perform the OFL Operation
Is Operation list empty?
no
Operations successful

The server generating the Image is able to check, by using UUID$_{OFLA}$, the availability of a pre-negotiated ARP from the OFL Authority for binding said Image with an OFL ARP state set to LOCKED in the OFL. PK.IMO$_{OFLA}$.ECDSA is first installed in the OFL using an Image containing no Image Segments. The OFL Authority is therefore a de facto an Image Owner. The OFL Authority has the ownership of the TRSH, thus controlling its usage by any Image Owners willing to load an Image to the OFL. This ARP filtering enforces the terms of a time constraint agreement sealed between the OFL Authority and an Image Owner.

The TRSH Maker may issue a specific ARP independently of the OFL Authority in order to perform OFL Operations related to Secure Hardware Platform Firmware. In that case, PK.TRS$_M$.ECDSA will be used, as shown in Figure 9-1.

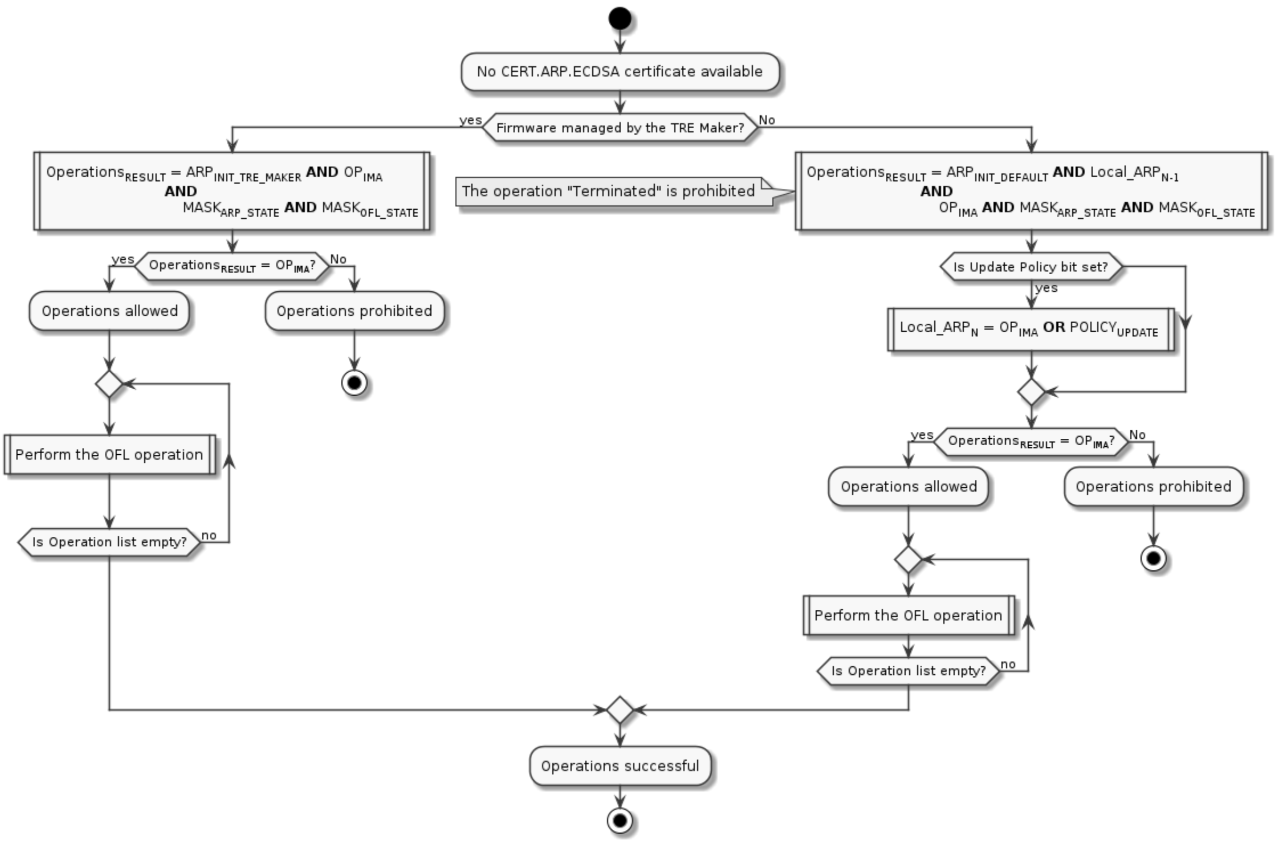
The selection of the ARP verification path (OFL Authority and TRSH Maker) is performed by using:

- The OFL Authority Identifier
- The TRSH Maker Authority Identifier

Both Identifiers are computed as defined in section 2.3.1.1 from the public key of the CA which has signed the ARP Certificate.

If no ARP Certificate is provided during the Image Loading, then the following procedure shall apply.

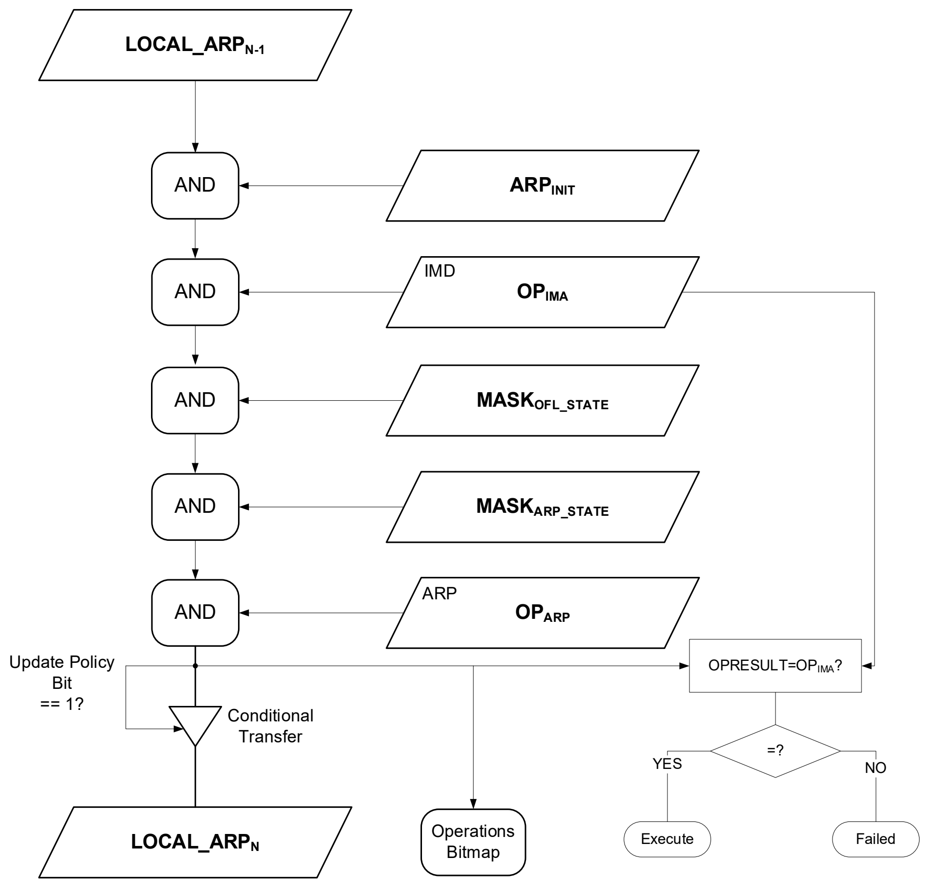
Figure 9-2: No ARP Certificate Use Case Processing


No CERT.ARP.ECDSA certificate available
Firmware managed by the TRE Maker?
yes
No
Operations_RESULT = ARP_INIT_TRE_MAKER AND OP_IMA AND MASK_ARP_STATE AND MASK_OFL_STATE
The operation "Terminated" is prohibited
Operations_RESULT = ARP_INIT_DEFAULT AND Local_ARP_N-1 AND OP_IMA AND MASK_ARP_STATE AND MASK_OFL_STATE
Operations_RESULT = OP_IMA?
Operations allowed
Operations prohibited
Is Update Policy bit set?
Local_ARP_N = OP_IMA OR POLICY_UPDATE
Perform the OFL operation
Is Operation list empty?
no
Operations successful

9.4 Procedure to Determine the Local ARP and the OFL Operations Bitmap

Figure 9-3 illustrates the procedure to determine the OFL operations bitmap and the local ARP.

Figure 9-3: Local ARP and OFL Operation Determination



$LOCAL\_ARP_N$ (next LOCAL_ARP) is stored in the Firmware Session.

The initial values of $LOCAL\_ARP_N$ are:

- **ARP_{INIT_OFLA}** for any Firmware Session from an Image Owner.
- **$ARP_{LOCAL\_ARP\_TRSM}$** for any Firmware Session from the TRSH Maker.

The logical AND between the $LOCAL\_ARP_{N-1}$ (current LOCAL_ARP), the OP_{IMA} in the IMD, the OP_{ARP}, and the MASK (see Table 7-4 and Table 7-5) is computed as shown in Figure 9-3. If the 'Update Policy' operation is requested, then the results are transferred into the LOCAL_ARP variable, which becomes $LOCAL\_ARP_N$. Only the bits related to the OFL Mask are updated in a persistent variable containing the LOCAL_ARP. The LOCAL_ARP is set to its initial value after each transfer for the OFL Authority.

The result is the logical AND for OP_{IMA}, MASK, and OP_{ARP} of the 16 least significant bits and the complement of the 16 most significant bits. The list of operations is defined in Table 9-1 to execute according to their priority order.

Operation result OP_{RESULT} **=** ARP_{INIT} **AND** $Local\_ARP_{N-1}$ **AND** OP_{IMA} **AND** OP_{ARP} **AND** $MASK_{ARP\_STATE}$ **AND** $MASK_{OFL\_STATE}$

Only the TRSH Maker is allowed to set the OFL to the TERMINATED state.

If the Update Policy bit is set in the operation result, then the following action is performed:

$$\text{Local\_ARP}_{\text{N-1}} = \text{Operations } \textbf{OR} \text{ POLICY}_{\text{UPDATE}}$$

If the Operation result $\text{OP}_{\text{RESULT}}$ is different than the OP_{IMA}, even by a single bit, then an error code shall be raised.

Only the value of the bits rank 6, 14, and 15 are persistent in the Firmware Session.

9.5 Procedure to Determine Precedence of OFL Operations

The OFL Operations bitmap defines the precedence of the OFL Operations for their execution.

OFL Operations shall be executed according to their priority order.

The encoding of the different OFL Operations into the requested operation 32-bit bitmap, as well as their respective priority, is defined in the following table.

Table 9-1: OFL Operation Priorities and Bitmap Coding

Operation	Operation Priority (0 lowest)	Bit Rank
Erase Firmware	2	1
Delete Firmware Session	3	2
Load Firmware	1	3
Disable Firmware	0	4
Enable Firmware	0	5
Backup/Restore Firmware Allowed	0	6
Lock OFL / Transfer Rights	5	7
RFU	5	8
Unlock OFL	5	9
Disable OFL	4	10
Enable OFL	4	11
Update Policy	7	12
Terminate OFL	8	13
Firmware Update Allowed	1	14
Local Firmware Delete Allowed	3	15
RFU1	5	16

The 16 most significant bits are the binary complement of the 16 least significant bits

Example: Erase Firmware, Load Firmware, & Enable Firmware = 'FFEA0015'

10 OFL Information

10.1 Overview

The OFL manages a set of information related to the Secure Hardware Platform and the OFL. The OFL Agent shall be able to read and write this set of information. The protocol supporting the reading/writing of this information is out of scope of this document and may be implementation dependent.

10.2 Secure Hardware Platform Identification

The Secure Hardware Platform securely and confidentially stores the following private data elements related to the identification of the Secure Hardware Platform. Table 10-1 defines the data used for the Secure Hardware Platform Identification:

Table 10-1: Secure Hardware Platform Identification Data Elements

Data Elements	Description
K_{BN}	The unique key of a Batch shared by the TRSH Maker HSM and by all Secure Hardware Platforms having the same Batch Number (BN). The size of the key shall be greater than or equal to 128 bits.
K_{PN}	The diversified key of the Secure Hardware Platform that is known by all Secure Hardware Platforms sharing the same Part Number (PN) and by the TRSH Maker HSM. The size of the key shall be greater than or equal to 128 bits.
SN	The Serial Number of the Secure Hardware Platform shall be unique within the scope of the TRSH Maker. SN is only known by the Secure Hardware Platform and is never provided in this form. SN is 64 bits long (padded if needed). SN shall be immutable.
BN	The Batch Number of the Secure Hardware Platform shall be unique within the scope of the TRSH Maker. BN is only known by the Secure Hardware Platform. BN is 64 bits long (padded if needed). BN shall be immutable.

The data elements in Table 10-1 shall be:

- Only known by the TRSH Maker and the Secure Hardware Platform; and
- Never changed during the lifetime of the TRSH.

The TRSH Maker HSM can decrypt the cryptogram by **K_{PN}** of BN padded with a random (randomization).

The OFL shall be able to generate an ephemeral universally unique and uniform Batch Number Pseudonym Identifier (BNPI) as a pseudonym of the Batch Number.

(BN) of Secure Hardware Platform.

BATCH_NUMBER as defined in Table 10-2 is a BNPI.

The ephemeral BNPI of BN appears as random and shall never be the same at each reading from the OFL registry.

BNPI = CIPHER (K_{PN}) [Rand|CATV|BN]

Where:

Rand	:	A 32-bit random number
CATV	:	A 32-bit secret constant for checking a successful decoding of BN
BN	:	The Batch Number
CIPHER	:	Defined in section 4.2.2

The TRSH Maker HSM can decrypt the BNPI by using the K_{PN}, verifies the CATV and the extracts the BN.

Only the OFL and the TRSH Maker HSM shall be able to extract the Batch Number (BN) from a BNPI. The extraction of BN from a BNPI by the TRSH Maker allows audit operations to be performed.

Rand|CATV|BN = ICIPHER (K_{PN}) [BNPI]

Where:

Rand	:	A random number
CATV	:	A 32-bit constant for checking a successful decoding of SN. CATV value is set to 'AA55AA55'.
BN	:	Batch Number of the Secure Hardware Platform
ICIPHER	:	Defined in section 4.2.2

The OFL shall be able to generate three type of universally unique and uniform Secure Hardware Platform Pseudonym Identifier (SHPPI) as pseudonyms of the serial number (SN) of Secure Hardware Platform:

- An Ephemeral SHPPI (EPH_PI). This identifier cannot be regenerated deterministically by OFL.
- A Deterministic SHPPI (DET_PI). This identifier can be regenerated deterministically by the OFL from a deterministic seed (URN) provided to the OFL.
- An Immutable SHPPI (IMU_PI). This identifier is a Deterministic SHPPI that requires no seed (immutable).

An Ephemeral SHPPI is a cryptogram of SN and shall not be the same at each reading from the OFL within the range of probability to get twice the random value called Rand. This SHPPI is computed as follows:

EPH_PI = CIPHER (K_{BN}) [Rand|CATV|SN]

Where:

Rand	:	A 32-bit random number
CATV	:	A 32-bit constant for checking a successful decoding of SN. CATV value is set to 'AA55AA55'.
SN	:	Serial Number of the Secure Hardware Platform
CIPHER	:	Defined in section 4.2.2

SERIAL_NUMBER is an Ephemeral SHPPI.

The OFL shall be able to generate a Deterministic SHPPI as follows:

DET_PI = CIPHER (K_{BN}) [HASH|CATV|SN]

Where:

HASH	:	The 32 least significant bits of a UUID version 5 from a URN
CATV	:	A 32-bit constant for checking a successful decoding of SN. CATV value is set to 'AA55AA55'.
SN	:	Serial Number of the Secure Hardware Platform
CIPHER	:	Defined in section 4.2.2

The OFL shall be able to generate an Immutable SHPPI as follows:

IMU_PI = CIPHER (K_{BN}) [0|CATV|SN]

Where:

0	:	'00000000'
CATV	:	A 32-bit constant for checking a successful decoding of SN. CATV value is set to 'AA55AA55'.
SN	:	Serial Number of the Secure Hardware Platform
CIPHER	:	Defined in section 4.2.2

The TRSH Maker HSM can decrypt the SHPPI by using the $\mathbf{K_{BN}}$, verify the CATV, and extract the serial number SN.

Only the OFL and the TRSH Maker HSM shall be able to extract the serial number SN from an SHPPI. The extraction of SN from an SHPPI by the TRSH Maker allows audit operations to be performed.

X|CATV|SN = ICIPHER (K_{BN}) [SHPPI]

Where:

X	:	'00000000' or HASH or Rand
CATV	:	A 32-bit constant for checking a successful decoding of SN. CATV value is set to 'AA55AA55'
SN	:	Serial Number of the Secure Hardware Platform
ICIPHER	:	Defined in section 4.2.2

10.3 OFL Registry

10.3.1 Description

Table 10-2 defines the OFL registry, which can be read by the OFL Agent.

The OFL Agent selects a Firmware using a Firmware UUID contained in the UUID_FIRMWARE_LIST registry entry and writes it in the UUID_FIRMWARE registry entry. Registry entries flagged with a dot in the FI column of Table 10-2 are filled by OFL with information related to the selected Firmware. If the UUID_FIRMWARE registry entry is not set to a Public Firmware Identifier listed in the registry entry UUID_FIRMWARE_LIST, then the corresponding registry entries shall be empty.

Table 10-2: OFL Registry

Parameter	Access Right	FI	Comment	Length	Default
UUID_FIRMWARE	RW	-	Public Identifier of a Firmware listed in the UUID_FIRMWARE_LIST registry entry. This Identifier allows the OFL Agent to determine to Firmware Session.	16	'FF'
OFL_VERSION	RO		Major and minor version of the Open Firmware Loader.	2	'0200'
SERIAL_NUMBER	RO		Serial Number of the Secure Hardware Platform (randomized from the SN) by using a diversified key related to PART_NUMBER and BATCH_NUMBER. SERIAL_NUMBER value is an EPH_PI and shall be different at each registry access.	16	-
PART_NUMBER	RO		Identifier of the Part Number as defined in section 2.3.	16	-
OFL_TYPE_UUID	RO		Open Firmware Loader Type Identifier as defined in section 2.3.	16	-
BATCH_NUMBER	RO		Ephemeral Batch Number Pseudonym Identifier (BNPI) of the Secure Hardware Platform as defined in section 10.2.	16	-
TRSH_CREDENTIALS _PARAMETER	RO		TRSH Credentials Parameter as defined in section 6.2.3.	-	-
IDS_CREDENTIALS _PARAMETER	WO		IDS Credentials Parameter as defined in section 6.2.2	-	-
OFL_STATE	RO		OFL states: • 01: ENABLED • 02: DISABLED • 03: TERMINATED	1	'01'

Parameter	Access Right	FI	Comment	Length	Default
ARP_STATE	RO		OFL ARP states: • 00: BLANK • 01: LOCKED • 02: UNLOCKED	1	-
FIRMWARE_STATE	RO		Firmware State: • 0 when the Firmware is in the ENABLED state. • 1 when the Firmware is in the DISABLED state.	1	-
LIST_ISID	RO		List of the Identifiers (ISID) of the pending Image Sessions.	N_1	-
UUID_FIRMWARE _LIST	RO		List of the public Identifiers of the Firmwares identifying the Firmware Sessions.	N_2*16	-
UUID_CI _GENERATION_LIST	RO		List of CI_{OFL} identified by their SKID available in the OFL. One of these CI shall be used for verifying the OFL Certification Path.	N_3*16	-
UUID_CI_ VERIFICATION_LIST	RO		List of CI_{OFL} and CI_{IDS} identified by their SKID available in the OFL. One of these CI shall be used for verifying the IDS Certification Paths.	N_4*16	-
OPERATION_TOKEN	RO		Contains the last stored Operation Token as defined in sections 5.4.5.1 and 5.4.5.3.	32	-
OPERATION_TOKEN _CHANGE	RO		Contains the last stored Operation Token as defined in section 5.4.5.2	N_5	-
BIST	RO		Contains the Build In Self-Test result. The implementation is proprietary and shall be left to the OFL provider.	1	-
GROUP_ID_LIST	RO		List of Firmware Group identifiers of all Firmware Sessions.	N_6*16	-
GROUP_ID	RO	●	Firmware Group identifier $UUID_G$ of the Firmware as assigned in the Firmware Session.	16	-
CURR_VERSION	RO	●	Current version (VER_CURR) of the Firmware in the Firmware Session	2	-
FIRMWARE_STATE	RO	●	Current state of the Firmware in the Firmware Session	1	
URN_PI	WO		URN for computing a deterministic SHPPI as defined in section 10.2.	N_7	-
DET_PI	RO		Deterministic SHPPI as defined in section 10.2	16	-

Parameter	Access Right	FI	Comment	Length	Default
EPH_PI	RO		Ephemeral SHPPI as defined in section 10.2	16	-
IMU_PI	RO		Immutable and deterministic SHPPI as defined in section 10.2	16	-
IDS2_EXT	RO		IDS2-Extensions in plaintext. This information is available after a successful Image Loading. This registry entry is cleared at the beginning of an Image Loading. The size of the registry entry is limited to 1024 bytes.	N_8	-
UUID_OFLA	RO		Identifier of the OFL Authority	16	-

Note: The IMU_PI shall be considered as personal data according to [GDPR] Article 4. The IMU_PI is provided to the OFL Agent without any protection as defined in [GDPR] Article 25. An OFL Agent that acquires the IMU_PI shall provide the means for protecting the IMU_PI according to [GDPR] Article 25.

10.3.2 OFL Registry Data Element

The OPERATION_TOKEN_CHANGE registry entry contains the following ASN.1 description encapsulating the Operation Tokens defined in section 5.4.5.2.

```
-- ASN1START
FirmwareState                         ::= ENUMERATED
{
eENABLED                              (0), -- Firmware in ENABLED state
eDISABLED                             (1)  -- Firmware in DISABLED state
}
OPERATION_TOKEN_CHANGE::= [APPLICATION 4] SEQUENCE
{
aOFL-Operation-Token                  OCTET STRING (SIZE(32)),
                                      --Operation Token as defined in section 5.4.5.2
aCounter                              Integer,
                                      --Counter
aState                                FirmwareState
                                      --Firmware state
}

-- ASN1STOP
```

Where:

aOFL-Operation-Token	:	Operation Token as defined in section 5.4.5.2		
aCounter	:	Counter as defined in section 5.4.5.2		
aState	:	Firmware state		
		eENABLED	:	0 when the Firmware is in ENABLED state
		eDISABLED	:	1 when the Firmware is in DISABLED state

11 Certificates and Authentication Tokens

11.1 Overview

11.1.1 Certification Paths

All Certificates received by the OFL and the IDS shall be validated.

To support disaster recovery, OFL Certification Paths should be backed up, in case a Certificate within the OFL Certification Path is revoked in the field.

The OFL PKI implements the X.509 Certificate format version 3 using DER encoding, as specified in [RFC 5280].

The OFL PKI and IDS PKI may support multiple Certificate Issuers (CI_{OFL} or CI_{IDS}).

The following Certification Paths shall be considered:

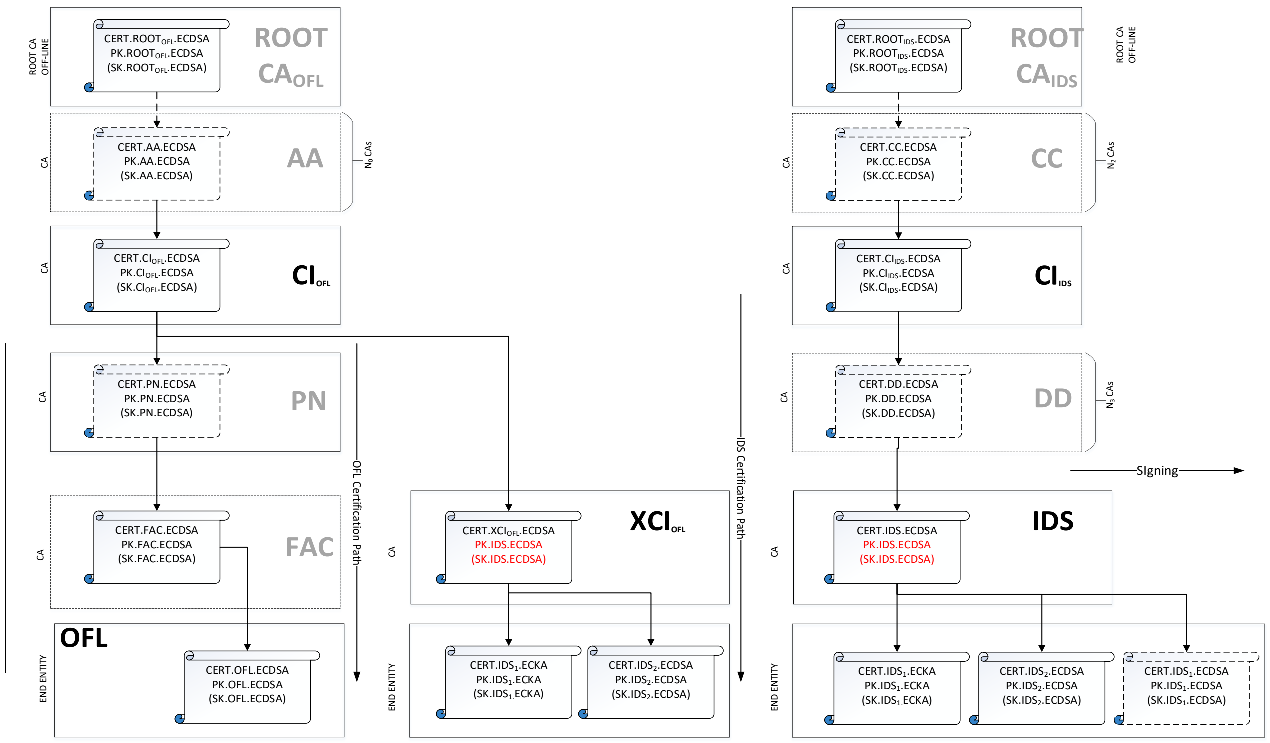
- PK.CI_{OFL}.ECDSA is the public key acting as a trust anchor for the following the OFL Certification Path:
  - PN CA (CERT.PN.ECDSA) is the CA and the FAC CA is its subject;
  - FAC CA (CERT.FAC.ECDSA) is the last CA of the OFL Certification Path;
  - OFL End Entity Certificate is a subject of the FAC CA.
- PK.CI_{OFL}.ECDSA is the public key acting as a trust anchor for the following IDS Certification Paths:
  - XCI_{OFL} CA (CERT. XCI_{OFL}.ECDSA) is the CA allowing the following Certification Paths:
    - The Certification Path to the End Entity Certificate (CERT.IDS_1.ECKA);
    - The Certification Path to the End Entity Certificate (CERT.IDS_2.ECDSA).
- PK.CI_{IDS}.ECDSA is the public key acting as a trust anchor for the following IDS Certification Paths:
  - IDS CA (CERT.IDS.ECDSA) is the last CA of the following Certification Paths:
    - The Certification Path to the End Entity Certificate (CERT.IDS_1.ECKA);
    - The Certification Path to the End Entity Certificate (CERT.IDS_2.ECDSA);
    - The Certification Path to the optional End Entity Certificate (CERT.IDS_1.ECDSA).

The following Certificates may be self-signed:

- IMO Certificate (CERT.IMO.ECDSA) used to enable the OFL Authority.

Figure 11-1 defines the Certification Paths.

Figure 11-1: Certification Paths

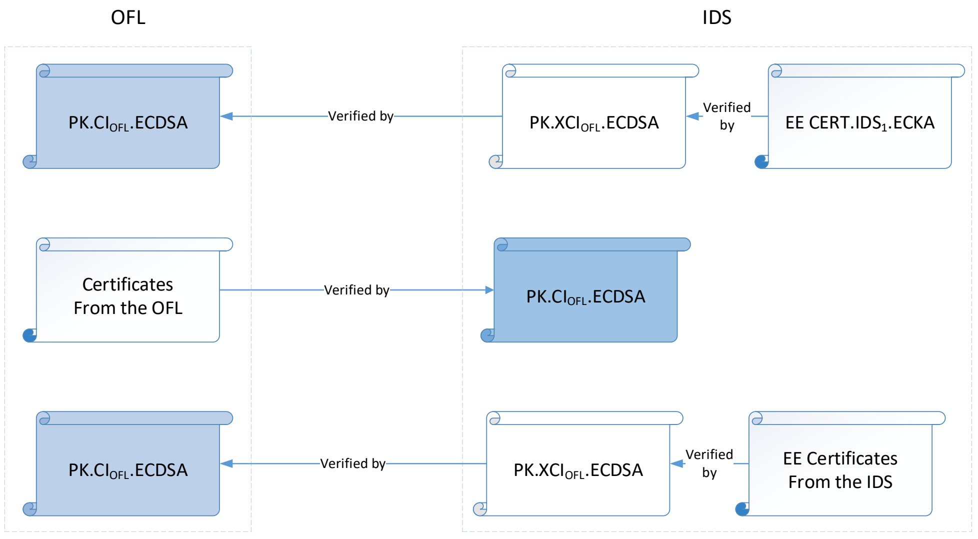

ROOT CA OFF-LINE
CERT.ROOT$_{OFL}$.ECDSA PK.ROOT$_{OFL}$.ECDSA (SK.ROOT$_{OFL}$.ECDSA)
ROOT CA$_{OFL}$
CA
CERT.AA.ECDSA PK.AA.ECDSA (SK.AA.ECDSA)
AA
N$_0$ CAs
CA
CERT.CI$_{OFL}$.ECDSA PK.CI$_{OFL}$.ECDSA (SK.CI$_{OFL}$.ECDSA)
CI$_{OFL}$
CA
CERT.PN.ECDSA PK.PN.ECDSA (SK.PN.ECDSA)
PN
CA
CERT.FAC.ECDSA PK.FAC.ECDSA (SK.FAC.ECDSA)
FAC
END ENTITY
OFL
CERT.OFL.ECDSA PK.OFL.ECDSA (SK.OFL.ECDSA)
OFL Certification Path
CA
CERT.XCI$_{OFL}$.ECDSA PK.IDS.ECDSA (SK.IDS.ECDSA)
XCI$_{OFL}$
END ENTITY
CERT.IDS$_1$.ECKA PK.IDS$_1$.ECKA (SK.IDS$_1$.ECKA)
CERT.IDS$_2$.ECDSA PK.IDS$_2$.ECDSA (SK.IDS$_2$.ECDSA)
IDS Certification Path
ROOT CA OFF-LINE
CERT.ROOT$_{IDS}$.ECDSA PK.ROOT$_{IDS}$.ECDSA (SK.ROOT$_{IDS}$.ECDSA)
ROOT CA$_{IDS}$
CA
CERT.CC.ECDSA PK.CC.ECDSA (SK.CC.ECDSA)
CC
N$_2$ CAs
CA
CERT.CI$_{IDS}$.ECDSA PK.CI$_{IDS}$.ECDSA (SK.CI$_{IDS}$.ECDSA)
CI$_{IDS}$
CA
CERT.DD.ECDSA PK.DD.ECDSA (SK.DD.ECDSA)
DD
N$_3$ CAs
Signing
CA
CERT.IDS.ECDSA PK.IDS.ECDSA (SK.IDS.ECDSA)
IDS
END ENTITY
CERT.IDS$_1$.ECKA PK.IDS$_1$.ECKA (SK.IDS$_1$.ECKA)
CERT.IDS$_2$.ECDSA PK.IDS$_2$.ECDSA (SK.IDS$_2$.ECDSA)
CERT.IDS$_1$.ECDSA PK.IDS$_1$.ECDSA (SK.IDS$_1$.ECDSA)

Several PKI configurations may occur for:

- PK.CI$_{IDS}$.ECDSA public key acting as a trust anchor for Certification Path ending at IDS$_1$ or IDS$_2$ End Certificates.
- PK.CI$_{OFL}$.ECDSA public key acting as a trust anchor for Certification Path ending at IDS$_1$ or IDS$_2$ End Certificates.

Figure 11-2 illustrates a PKI configuration for which all Certification Paths originate from PK.CI$_{OFL}$.ECDSA.

Figure 11-2: Certification Path Verification from CI$_{OFL}$ Trust Anchor

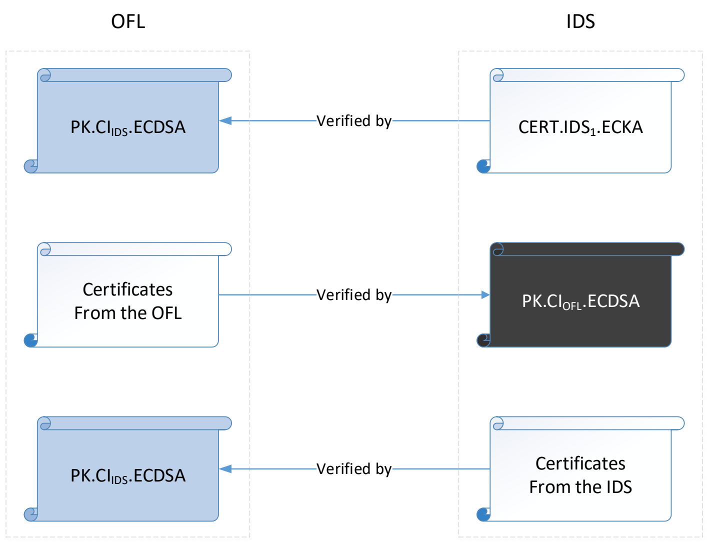


The verification of the IDS Certification Path by the OFL is allowed by provisioning CERT.XCI$_{OFL}$.ECDSA into the IDS so that it can be returned during the communication between the IDS and the OFL.

The verification of the OFL Certification Path by the OFL is allowed by provisioning PK.CI$_{OFL}$.ECDSA into the IDS.

Figure 11-3 illustrates a PKI configuration for which the OFL Certification Paths originate from PK.CI$_{OFL}$.ECDSA and the IDS Certification Paths originate from PK.CI$_{IDS}$.ECDSA, as defined in Figure 11-1.

Figure 11-3: Certification Path Verification from CI$_{IDS}$ and CI$_{OFL}$ Trust Anchors



The verification of the IDS Certification Path by the OFL is allowed by provisioning the trust anchor as PK.CI$_{IDS}$.ECDSA public key into the OFL.

The verification of the OFL Certification Path by the OFL is allowed by provisioning the trust anchor as PK.CI$_{OFL}$.ECDSA public key into the IDS.

11.1.2 Certification Paths Verification

11.1.2.1 Certification Path Validation Procedure

Each Certificate shall have a subject key identifier (SKID).

Each Certificate shall have an Authority Key identifier (AKID).

The Certificate path verification algorithm is the following:

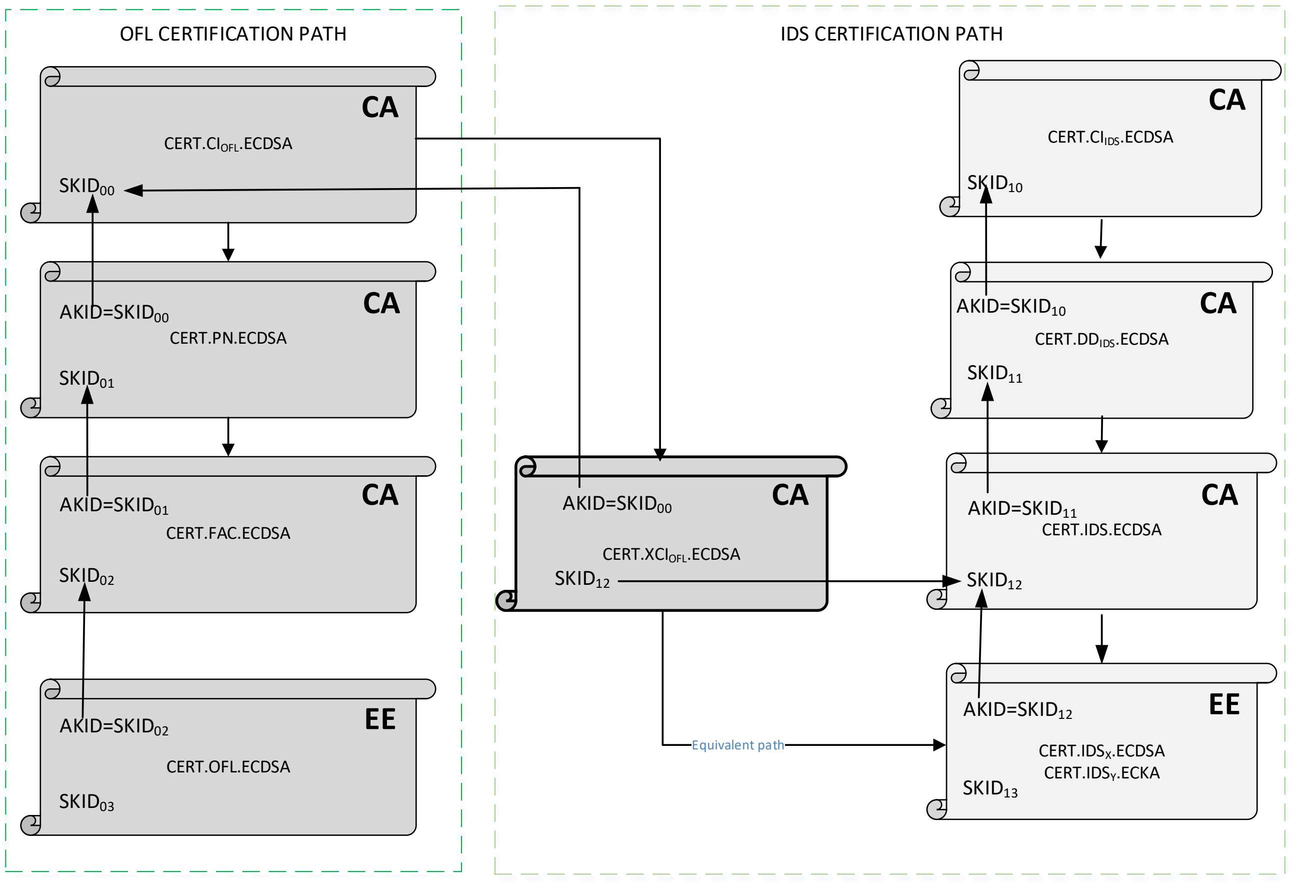
- The public key algorithm and parameters shall be checked.
- All public key algorithm and parameters for digital signature shall be equal in the Certificate path.
- The signature on the Certificate shall be verified using public key algorithm, the public key, and the public key parameters of the previous Certificate in the path.
- The date/time validity of the Certificate in the Certification Path shall be checked if the trusted time stamp (TIME-STAMP) defined in section 11.3.4 is provided.
- The IDS may check the Certificate revocation status of the OFL Certification Path by using OCSP.
- The authority key identifier (AKID) of a Certificate shall be checked to ensure that it equals the subject key identifier(SKID) of its CA.

- The asserted Certificate Policy OIDs are checked against the permissible OIDs as of the previous Certificate, including any policy mapping equivalencies asserted by the previous Certificate.
- Policy constraints are checked to ensure that any explicit policy requirements are not violated; basic constraints are checked to ensure that the Certificate is a CA Certificate:
  - The path length is checked to ensure that it does not exceed any maximum path length asserted in the current or a previous Certificate;
  - The key usage extension is checked to ensure that is allowed to sign Certificates; and
  - Any other critical extensions are recognized and processed.

If this procedure reaches the last Certificate in the Certification Path, with no name constraint or policy violations or any other error condition, then the Certification Path validation algorithm terminates successfully.

Figure 11-4 illustrates all Certification Path configurations.

Figure 11-4: Certification Path Configurations



Note: The XCI$_{OFL}$ Certificate is issued by the CI$_{OFL}$ in order to ensure that the use of OFL is still possible even if the TRSH Maker or the Device Maker no longer exists (e.g. due to bankruptcy). The CI$_{OFL}$ should be public and widely recognized as a trust party.

11.1.2.2 Certification Path Verification from CI_{OFL} Trust Anchor

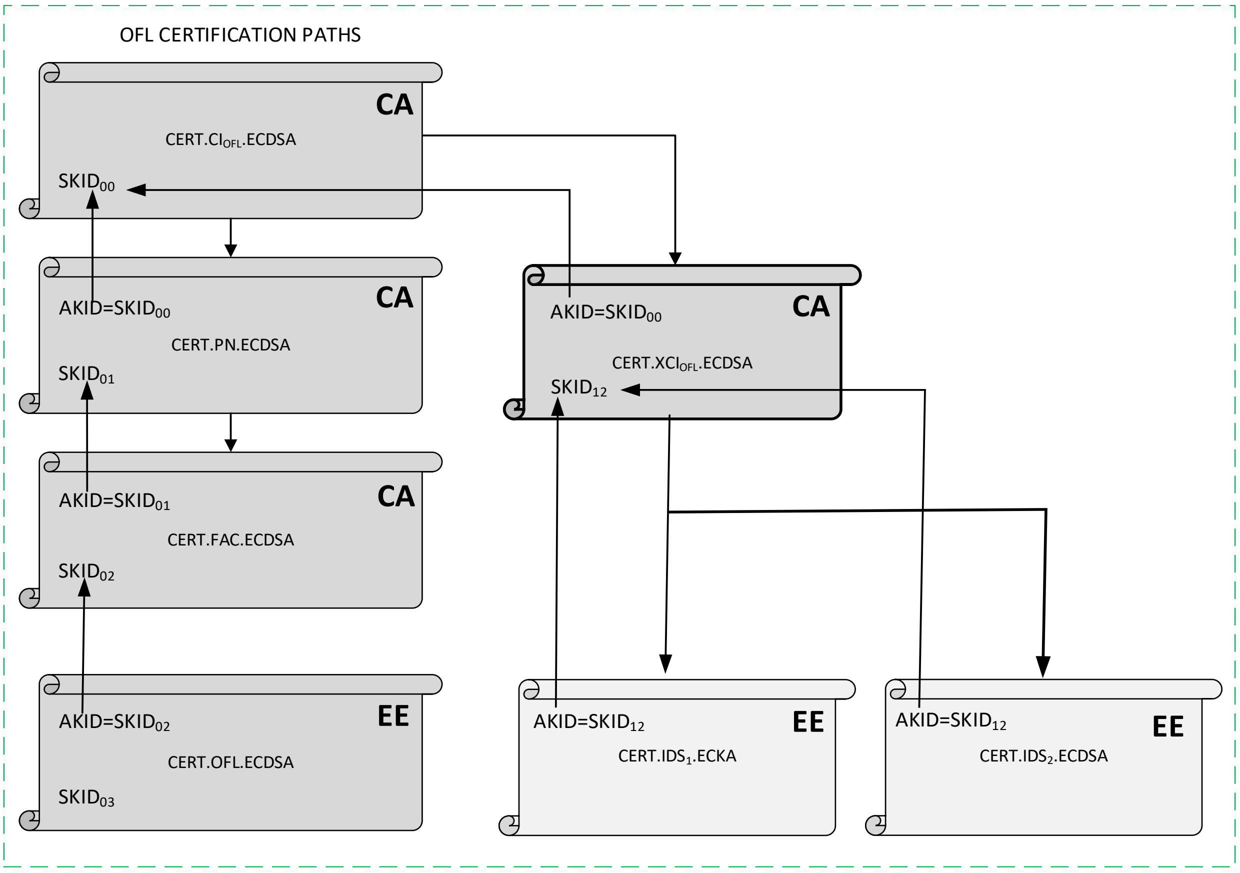
The OFL shall accept the XCI_{OFL} Certificate allowing to verify the End Entity Certificate of the IDS Certification Paths. All Certification Path verification is performed by using the PK.CI_{OFL}.ECDSA public key.

The CERT.XCI_{OFL}.ECDSA Certificate is issued by the CI_{OFL} CA to the last CA of the IDS Certification Path (CERT.IDS.ECDSA). The IDS provides this XCI_{OFL} Certificate (CERT.XCI_{OFL}.ECDSA) at each connection of the OFL.

Figure 11-5 is given as example for illustrating CERT.XCI_{OFL}.ECDSA Certificate signed by the CI_{OFL} CA and allowing the End Entity (EE) Certificates ending an IDS Certification Path to be originated by the PK.CI_{OFL}.ECDSA public key.

Figure 11-5 illustrates the verification of the End Entity Certificate of IDS Certification Paths (CERT.IDS_1.ECKA and CERT.IDS_2.ECDSA).

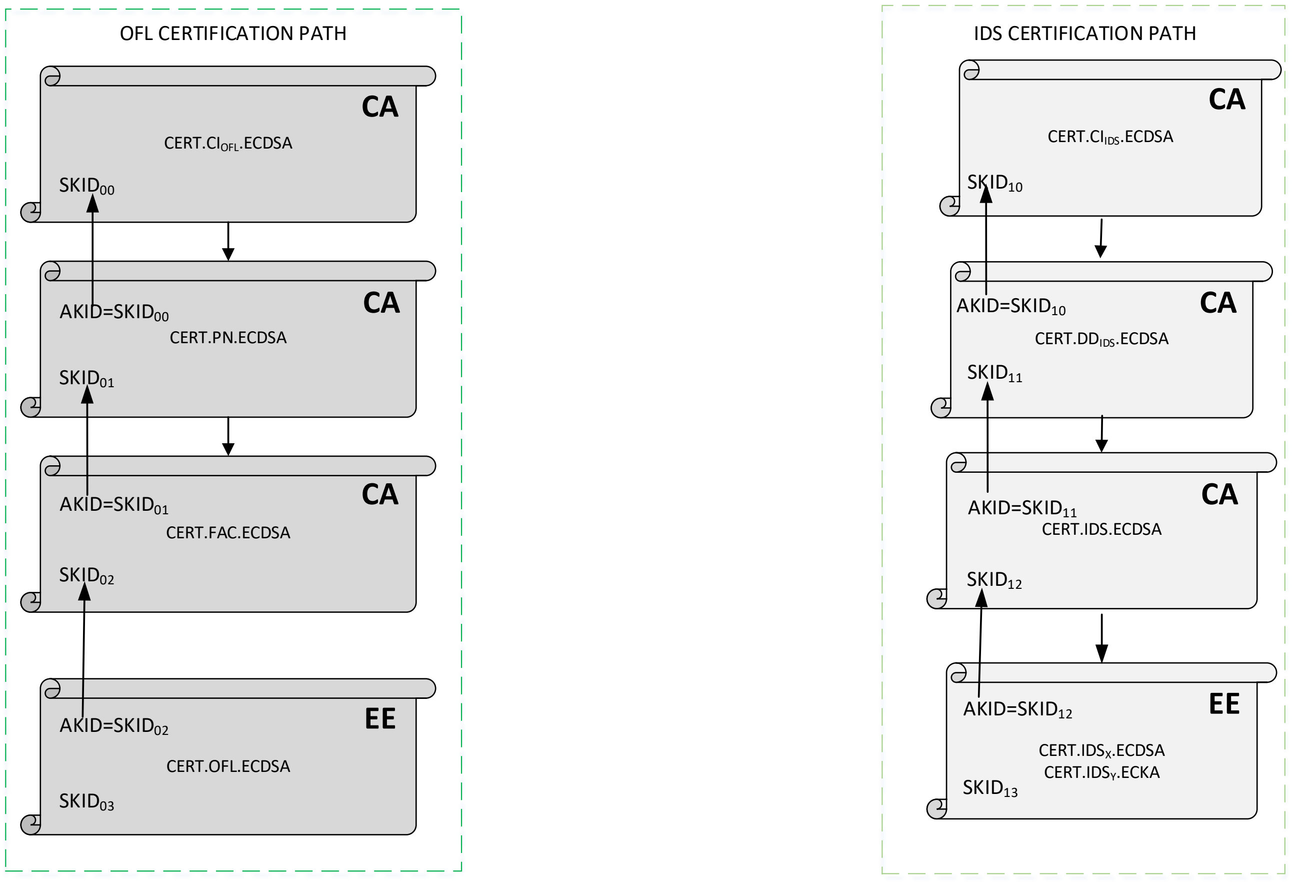
Figure 11-5: Certification Path Verification from CI_{OFL} Trust Anchor



11.1.2.3 Certification Paths Verification from CI_{OFL} and CI_{IDS} Trust Anchors

Figure 11-6 illustrates the IDS and the OFL Certification Paths requiring the provisioning of the trust anchors $PK.CI_{OFL}.ECDSA$ and the $PK.CI_{IDS}.ECDSA$ public keys respectively to the IDS and the OFL.

Figure 11-6: Certification Path Verification from CI_{OFL} and CI_{IDS} Trust Anchors



11.1.3 Certificate Requirements

11.1.3.1 Algorithm Identifiers and Parameters

This section specifies the values to be set in the 'AlgorithmIdentifier.algorithm' and 'AlgorithmIdentifier.parameters' fields of the Certificate for each algorithm used in this specification.

For section 'subjectPublicKeyInfo', the following settings shall apply:

- 'AlgorithmIdentifier.algorithm' field shall be set to: "iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) ecPublicKey(1)" as defined in [RFC 5480].
- 'AlgorithmIdentifier.parameters' field shall be set to:
  - for BrainpoolP256r1: "iso(1) identified-organization(3) teletrust(36) algorithm(3) signatureAlgorithm(3) ecSign(2) ecStdCurvesAndGeneration(8) ellipticCurve(1) versionOne(1) brainpoolP256r1(7)" as defined in [RFC 5639]
  - for BrainpoolP384r1: “iso(1) identified-organization(3) teletrust(36) algorithm(3) signatureAlgorithm(3) ecSign(2) ecStdCurvesAndGeneration(8) ellipticCurve(1) versionOne(1) brainpoolP384r1(11)”
  - for NIST P-256: "iso(1) member-body(2) us(840) ansi-X9-62(10045) curves(3) prime(1) prime256v1(7)" as defined in [RFC 5480]
  - for NIST P-384: “iso(1) identified-organization(3) certicom(132) curve(0) ansip384r1(34)
  - for FRP256V1: "iso(1) member-body(2) fr(250) type-org(1) 223 101 256 1" as defined in [ANSSI ECC]

For sections 'signature' and 'signatureAlgorithm', the following settings shall apply:

- 'AlgorithmIdentifier.algorithm' field shall be set to: "iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)" as defined in [RFC 5758] and [RFC 5759]
- 'AlgorithmIdentifier.parameters' field shall be omitted as defined in [RFC 5758] section 3.2.

11.1.3.2 Certificate Policy OID

Definition of OIDs for role identification using the GlobalPlatform root node 1.2.840.114283:

```
-- ASN1START
id-globalplatform  OBJECT IDENTIFIER ::={ iso(1) member-body(2) us(840) globalplatform(114283)}

id-ofl-role        OBJECT IDENTIFIER ::={ id-globalplatform ofl(10) role(1)}
id-role-imo        OBJECT IDENTIFIER ::={ id-ofl-role imo(1)}
id-role-arp        OBJECT IDENTIFIER ::={ id-ofl-role arp(2)}
id-role-ci-ofl     OBJECT IDENTIFIER ::={ id-ofl-role ci(3)}
id-role-ci-ids     OBJECT IDENTIFIER ::={ id-ofl-role ci(4)}
id-role-xci        OBJECT IDENTIFIER ::={ id-role-ci-ofl xci(1)}
id-role-pn         OBJECT IDENTIFIER ::={ id-ofl-role pn(3)}
id-role-fac        OBJECT IDENTIFIER ::={ id-role-pn  fac(1)}
id-role-ofl        OBJECT IDENTIFIER ::={ id-role-fac ofl(1)}
id-role-dd         OBJECT IDENTIFIER ::={ id-role-ci-ids dd(1) }
id-role-ids-asym   OBJECT IDENTIFIER ::={ id-role-dd  ids(1) }
id-role-ids-sym   OBJECT IDENTIFIER  ::={ id-role-xci    ids(1) }
id-role-ids1-sym-ecdsa OBJECT IDENTIFIER ::={ id-role-ids-sym   ids1-ecdsa(1) }
id-role-ids1-sym-ecka  OBJECT IDENTIFIER ::={ id-role-ids-sym   ids1-ecka(2) }
id-role-ids2-sym       OBJECT IDENTIFIER ::={ id-role-ids-sym   ids2(3)}
id-role-ids1-asym-ecdsa OBJECT IDENTIFIER ::={ id-role-ids-asym   ids1-ecdsa(1) }
id-role-ids1-asym-ecka  OBJECT IDENTIFIER ::={ id-role-ids-asym   ids1-ecka(2) }
id-role-ids2-asym       OBJECT IDENTIFIER ::={ id-role-ids-asym   ids2(3)}

-- ASN1STOP
```

11.1.3.3 Basic Constraints

The IDS and FAC Certificates respectively CERT.IDS.ECDSA and CERT.FAC.ECDSA shall include the following basic constraint:

- id-ce-basicConstraints OBJECT IDENTIFIER ::= { id-ce 19 }
- cA = true
- pathLenConstraint = 0

11.1.3.4 Certificate Revocation

11.1.3.4.1 Common Requirements

All online CA may provide an OCSP interface as defined in [RFC 6960]. All CA Certificates may include the Authority Information Access according to [RFC 5280] section 4.2.2.1.

When included in a Certificate, the Authority Information Access Extension shall contain the AccessDescription identified by the object identifier id-ad-ocsp (OBJECT IDENTIFIER ::= { id-ad 1 }) and containing the URL of the OCSP interface.

All CA Certificates shall be revocable individually.

End Entity Certificates shall not be revocable individually.

11.1.3.4.2 Certification Paths Configuration from $\mathrm{CI}_{\mathrm{OFL}}$ Trust Anchor

The revocation status of all Certificates in the IDS Certification Path by the OFL is globalized by the verification of the $\mathrm{XCI}_{\mathrm{OFL}}$ Certificate (CERT.$\mathrm{XCI}_{\mathrm{OFL}}$.ECDSA).

The renewal of the $\mathrm{XCI}_{\mathrm{OFL}}$ Certificate shall be performed on a periodic basis defined by a Certificate Policy (e.g. based on [RFC 3647]) from the $\mathrm{CI}_{\mathrm{OFL}}$ CA and may use an interface based on CMP as defined in [RFC 4210].

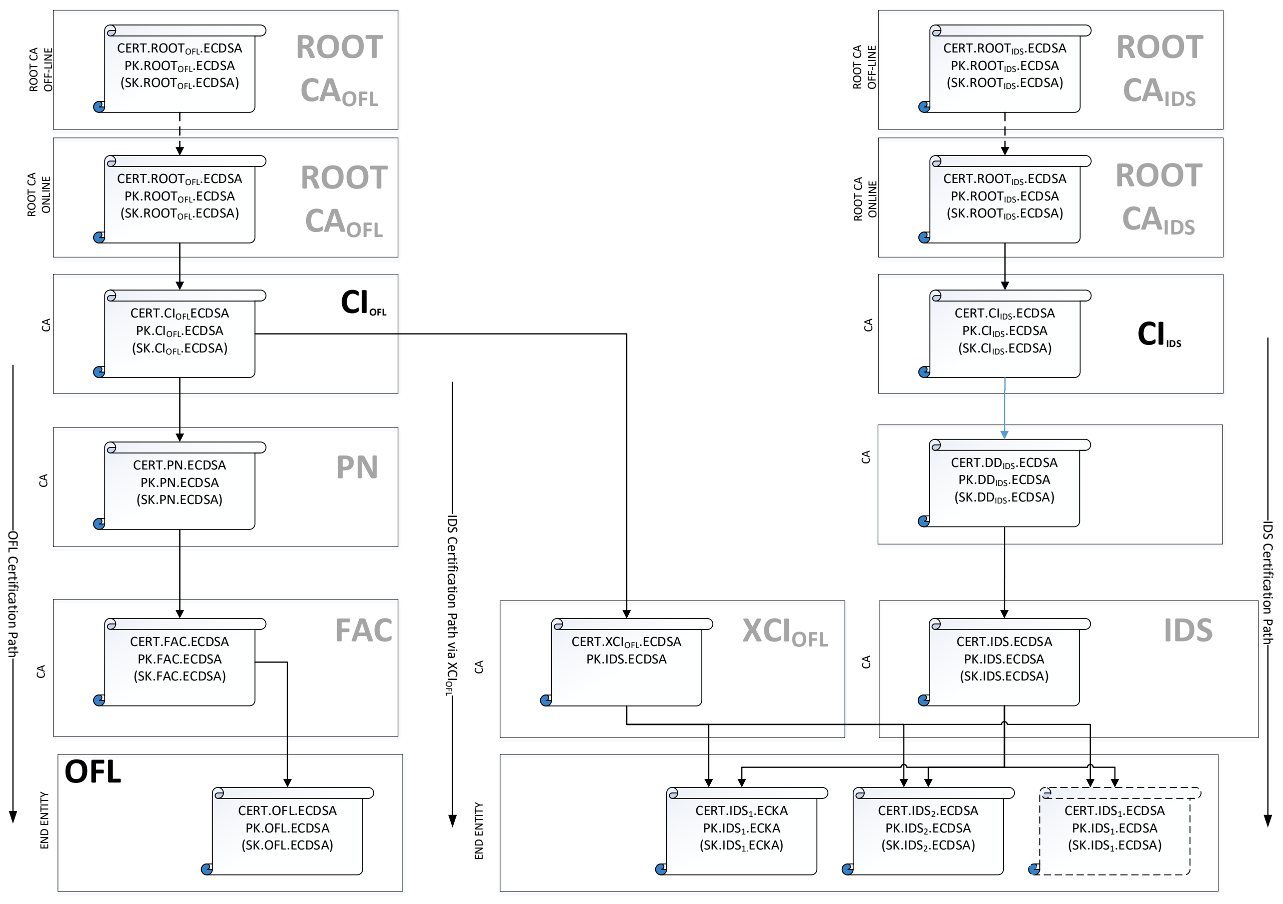
11.1.3.4.3 Certification Paths Configuration from $\mathrm{CI}_{\mathrm{OFL}}$ and $\mathrm{CI}_{\mathrm{IDS}}$ Trust Anchors

The revocation status of all Certificates in the IDS Certification Path by the OFL shall be performed by verifying the OCSP responses from all CA of the IDS Certification Path. For each CA, the public key of trusted responder as defined in [RFC 6960] section 2.2 shall be equal to the public key used for the Certificate signature generation.

11.1.4 Minimal OFL PKI

Figure 11-7 defines the Certification Paths for a minimal OFL PKI. All End Entity Certificates in the IDS and OFL Certification Paths are validated from the PK.CI$_{OFL}$.ECDSA public key.

Figure 11-7: Minimal Certification Paths



The CERT.XCI$_{OFL}$.ECDSA Certificate is equivalent to CERT.IDS.ECDSA. Verifying CERT.XCI$_{OFL}$.ECDSA using PK.CI$_{OFL}$.ECDSA public key is equivalent to verify the CERT.IDS.ECDSA Certificate using CERT.CI$_{IDS}$.ECDSA. The IDS Certification Paths originating from the PK.CI$_{OFL}$.ECDSA public key contains the following Certificates:

- CERT.XCI$_{OFL}$.ECDSA and CERT.IDS$_2$.ECDSA,
- CERT.XCI$_{OFL}$.ECDSA and CERT.IDS$_1$.ECKA.

11.2 Certificates

11.2.1 CI Certificates

The CI Certificates CERT.CI$_{\text{OFL}}$.ECDSA and CERT.CI$_{\text{IDS}}$.ECDSA are X.509 CA Certificates. The OFL may support multiple CI$_{\text{OFL}}$ and CI$_{\text{IDS}}$ as Certificate Issuers. CI$_{\text{OFL}}$ and CI$_{\text{IDS}}$ Certificate Issuers may be the same or may be different.

CERT.CI$_{\text{OFL}}$.ECDSA and CERT.CI$_{\text{IDS}}$.ECDSA are CA Certificates and may be cross-certificates as defined in [RFC 5280] section 3.2.

If the OFL supports multiple CI$_{\text{OFL}}$, then the OFL shall host as many Certificates as the ones included in the Certification Path, defined in section 11.1.1, for each supported CI$_{\text{OFL}}$.

Table 11-1: Certificate Issuer Certificate

Field	**Value Description**				
tbsCertificate	Data to be signed				
	Field		**Value Description**		
	Version		Shall be version 3 (value is 2) as extensions are used in this specification.		
	serialNumber		Shall be unique for each Certificate issued with a given CERT.CI$_{\text{OFL}}$.ECDSA or CERT.CI$_{\text{IDS}}$.ECDSA.		
	signature		Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'.		
	Issuer		Distinguished Name of the CI$_{\text{OFL}}$ that has signed this Certificate.		
	Validity		Validity period of the Certificate.		

	subject		Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
			Example of a DN:
			`c = US`
			`l = New York`
			`o = ACME`
			`cn = ACME IDS`
			`e = ids@acme.com`		

	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Shall be equal to the content of the Signature field.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	AKID of the Certificate		

Field	Value Description				
		id-ce-subjectKeyIdentifier	Subject Key Identifier		
		false			
		keyIdentifier [0]	Contains the SKID of the Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Sign all Certificates of the CI		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ci-ofl id-role-ci-ids	See section 11.1.3.2. May contain one or both OID (id-role-ci-ofl, id-role-ci-ids)		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.2 XCI$_{OFL}$ Certificate

11.2.2.1 Description

The XCI$_{OFL}$ Certificate belongs to the Certification Paths originating from PK.CI$_{OFL}$.ECDSA and ending to the End Entity IDS$_1$ and IDS$_2$ Certificates (CERT.IDS$_1$.ECKA and CERT.IDS$_2$.ECDSA). A valid Certification Path including the XCI$_{OFL}$ Certificate may be restricted to a given Part Number and/or Firmware Family Identifiers and may define additional condition of validation via optional extensions.

The XCI$_{OFL}$ Certificate is a CA cross-certificate as defined in [RFC 5280] section 3.2.

The following table describes the XCI$_{OFL}$ Certificate data related to CERT.XCI$_{OFL}$.ECDSA Certificate.

Table 11-2: XCI$_{OFL}$ Certificate Data

Field	Value Description	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued with a given CERT.CI$_{OFL}$.ECDSA.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.
	issuer	Distinguished Name of the CI$_{OFL}$ that has signed this Certificate.
	validity	Validity period of the Certificate. This End of validity period shall be daily updated after the checking of the revocation status of its subject Certificate.
	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes. Example of a DN: `c = US` `l = New York` `o = ACME` `cn = ACME IDS` `e = ids@acme.com`
	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.

Field	Value Description				
				Reference	Ref Section
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	Identify the PK.CI$_{OFL}$.ECDSA use for verifying this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the SKID of the IDS Certificate (CERT.IDS.ECDSA).		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Digital signature of the XCI Certificate		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-xci	See section 11.1.3.2		
		id-ofl-xci-extension	XCI$_{OFL}$ Extension	OFL 2.0	11.2.9.2
		true			
		XCIExtension	See section 11.2.2.2.		
		id-ce-basicConstraints	Basic Constraints	[RFC 5280]	4.2.1.9
		true			
		BasicConstraints	CA = true and pathLenConstraint = 0		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.2.2 XCI$_{OFL}$ Certificate Extension

The following ASN.1 description defines the constraints for XCI$_{OFL}$ use.

```
-- ASN1START
id-ofl-xci-extension                   OBJECT IDENTIFIER ::={ id-globalplatform ofl(10) role(2)}

PART-NUMBER                            ::= [PRIVATE 17]UUID
                                       --  Part Number Identifier


PathConstraint                         ::=CHOICE
{
aFirmwareFamilyID-Descriptor           FirmwareFamilyID-Descriptor,
                                       -- Firmware Family Identifier
aPNID                                  PART-NUMBER
                                       --Part Number Identifier

}

FirmwareFamilyID-Descriptor            ::=SEQUENCE
{
aNID                                   IA5String,
                                       --URN: Namespace Identifier associated to the Firmware Family
Identifier
aNSS                                   IA5String,
                                       --URN: Name Specific String of the Firmware Family Identifier
aFirmwareFamilyId                      UUIDF
                                       --Firmware Family Identifier as defined in OFL 2.0 section 2.3.
}
XCIExtension                           ::= SEQUENCE
{
aFFIS                                  SEQUENCE OF PathConstraint
}
-- ASN1STOP
```

Where:

aNID	:	The Namespace Identifier defined in section 2.3
aNSS	:	The Name Specific String defined in section 2.3
aFirmwareFamilyId	:	The Firmware Family Identifier defined in section 2.3. If present, the rules defined in section 8.2 apply.
aPNID	:	The Part Number Identifier, defined in section 2.3, for which the XCI$_{OFL}$ certificate is valid. If this parameter is absent, then the XCI$_{OFL}$ certificate is valid for any Part Number Identifier.

11.2.3 Distributed Delivery Certificate

11.2.3.1 Description

The Distributed Delivery CA shall be the first CA of the IDS Certification Path originated by the PK.CI$_{\text{IDS}}$.ECDSA public key.

This Certificate is a CA Certificate as defined in [RFC 5280] section 3.2.

The presence of this CA in the Certification Path is optional and dependent on a risk management decision linked with the number of IDS to manage.

The following table describes the Distributed Delivery Certificate data related to CERT.DD$_{\text{IDS}}$.ECDSA Certificate.

Table 11-3: Distributed Delivery Certificate Data

Field	Value Description	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	Version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued with a given CERT.CI$_{\text{IDS}}$.ECDSA.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{\text{IDS}}$.ECDSA defined in Table 11-1.
	Issuer	Distinguished Name of the CI$_{\text{IDS}}$ that has signed this Certificate.
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{\text{IDS}}$.ECDSA defined in Table 11-1.

Field	Value Description				
				Reference	Ref Section
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	Identify the PK.CI$_{IDS}$.ECDSA that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the identifier of the public key of the DD Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Digital Signature		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-dd	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.4 IDS Certificate

The IDS CA shall be the last CA of the IDS Certification Path originated by the PK.CI$_{IDS}$.ECDSA public key.

The following table describes the IDS Certificate data related to CERT.IDS.ECDSA Certificate.

Table 11-4: IDS Certificate Data

Field	**Value Description**	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued with a given CI$_{IDS}$ CA or DD CA.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{IDS}$.ECDSA defined in Table 11-1.
	issuer	Distinguished Name of the CI$_{IDS}$ that has signed this Certificate.
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{IDS}$.ECDSA defined in Table 11-1.

Field	**Value Description**				
				Reference	**Ref Section**
		id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	Identify the PK.CI$_{IDS}$.ECDSA or PK.DD.ECDSA that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
	Extensions	keyIdentifier [0]	Contains the identifier of the public key of the IDS Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Digital Signature		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ids-sym, id-role-ids-asym	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.5 IDS$_1$ Certificate for Key Agreement

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

Table 11-16 describes the Certificate content of the CERT.IDS$_1$.ECKA.

Table 11-5: IDS$_1$ Certificate Format

Field	**Value Description**				
tbsCertificate	Data to be signed				
	Field		**Value Description**		
	version		Shall be version 3 (value is 2) as extensions are used in this specification.		
	serialNumber		Shall be unique for each Certificate issued with a given CERT.IDS.ECDSA.		
	signature		Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA.		
	issuer		Distinguished Name of the IDS CA that has signed this Certificate.		
	validity		Validity period of the Certificate.		

	subject		Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
			Example of a DN:
			`c = US`
			`l = New York`
			`o = ACME`
			`cn = ACME IDS`
			`e = ids@acme.com`		

	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID of the PK.IDS.ECDSA that has to be used to verify this Certificate.		

Field	Value Description				
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the SKID of this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyAgreement (4)	Key Agreement between OFL and HSM_IDS$_1$		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ids1-sym-ecka, id-role-ids1-asym-ecka	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.6 IDS$_1$ Certificate for Digital Signature

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

The following table describes the Certificate data related to CERT.IDS$_1$.ECDSA. The CERT.IDS$_1$.ECDSA Certificate shall be signed by using SK.IDS.ECDSA.

Table 11-6: IDS$_1$ ECDSA X.509 Certificate Format

Field	Value Description				
tbsCertificate	Data to be signed				
	Field		**Value Description**		
	version		Shall be version 3 (value is 2) as extensions are used in this specification.		
	serialNumber		Shall be unique for each Certificate issued by the IDS CA (CERT.IDS.ECDSA).		
	signature		Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA Certificate.		
	issuer		Distinguished Name of the IDS CA that has signed this Certificate.		
	validity		Validity period of the Certificate.		

	subject		Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
			Example of a DN:
			`c = US`
			`l = New York`
			`o = ACME`
			`cn = ACME IDS`
			`e = ids@acme.com`		

	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{IDS}$.ECDSA defined in Table 11-1.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	**Authority Key Identifier**	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID of the PK.IDS.ECDSA that has to be used to verify this Certificate.		

Field	**Value Description**				
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the identifier of the public key bound in this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Sign the ATK.IDS_1.ECKA Authentication Tokens.		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ids1-sym-ecdsa, id-role-ids1-asym-ecdsa	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.7 IDS$_2$ Certificate for Digital Signature

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

The following table describes the Certificate data related to CERT.IDS$_2$.ECDSA. The CERT.IDS$_2$.ECDSA Certificate shall be signed by using SK.IDS.ECDSA.

Table 11-7: IDS$_2$ ECDSA X.509 Certificate Format

Field	**Value Description**				
tbsCertificate	Data to be signed				
	Field		**Value Description**		
	version		Shall be version 3 (value is 2) as extensions are used in this specification.		
	serialNumber		Shall be unique for each Certificate issued by the IDS CA (CERT.IDS.ECDSA).		
	signature		Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA.		
	issuer		Distinguished Name of the IDS CA that has signed this Certificate.		
	validity		Validity period of the Certificate.		

	subject		Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
			Example of a DN:
			`c = US`
			`l = New York`
			`o = ACME`
			`cn = ACME IDS`
			`e = ids@acme.com`		

	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	**Authority Key Identifier**	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID of the PK.IDS.ECDSA that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2

Field	**Value Description**				
		false			
		keyIdentifier [0]	SKID of this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Sign the ATK.IDS$_2$.ECKA Authentication tokens.		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ids2-sym-ecdsa, id-role-ids2-asym-ecdsa	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.8 OFL Certificate

The following table describes the Certificate data related to CERT.OFL.ECDSA Certificate.

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

The CERT.OFL.ECDSA Certificate shall be signed by using SK.FAC.ECDSA.

Table 11-8: OFL ECDSA X.509 Certificate Format

Field	Value Description				
tbsCertificate	Data to be signed				
	Field		**Value Description**		
	version		Shall be version 3 (value is 2) as extensions are used in this specification.		
	serialNumber		Shall be unique for each Certificate issued by a given FAC CA.		
	signature		Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.		
	issuer		Distinguished Name of the FAC Certificate that has signed this Certificate.		
	validity		Validity period of the Certificate. There is no life time defined for this Certificate. OFL Certificates never expire. Expiration Date to be set to 99991231235959Z as stated in [RFC 5280].		

	subject		Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
			It shall be in the same Organization Unit (OU) as the FAC Certificate.
			Example of a DN:
			`c = US`
			`l = New York`
			`o = ACME`
			`cn = ACME IDS`
			`e = ids@acme.com`		

	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			

Field	Value Description				
		keyIdentifier [0]	SKID of the PK.FAC.ECDSA public key that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	SKID of this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Sign the ATK.OFL.ECKA Authentication Tokens.		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-ofl	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.9 Part Number Certificate

11.2.9.1 Description

There shall be a single Part Number CA per Part Number and Certification Path, irrespective of the number of equipment (HSM) issuing the OFL Certificates.

This Certificate is a CA Certificate as defined in [RFC 5280] section 3.2.

The following table describes the PN (Part Number) Certificate data related to CERT.PN.ECDSA Certificate.

Table 11-9: PN ECDSA X.509 Certificate Format

Field	**Value Description**	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued with a given CERT.CI$_{OFL}$.ECDSA.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.
	issuer	Distinguished Name of the CI$_{OFL}$ that has signed this Certificate.
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.

Field	Value Description				
				Reference	Ref Section
		id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID of the PK.CI$_{OFL}$.ECDSA public key that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
	Extensions	keyIdentifier [0]	SKID of this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Digital signature of the OFL Certificate		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-pn	See section 11.1.3.2		
		id-part-number	PN Extension	OFL 2.0	11.2.9.2
		true			
		PartNumberExtension	See section 11.2.9.2.		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.9.2 Part Number Extension

The following ASN.1 description defines the Part Number extension.

```
-- ASN1START
id-part-number        OBJECT IDENTIFIER ::={ id-globalplatform ofl(10) pn(1)}

PartNumberExtension                    ::= SEQUENCE
{
aNID                                   IA5String,
                                       --URN: Namespace Identifier of the TRSH Maker
aNSS                                   IA5String,
                                       --URN: Name Specific String of the Part Number
aPartNumberId                          PART-NUMBER
                                       --Part Number Identifier as defined in section 2.3.

}
-- ASN1STOP
```

The IDS shall verify that the NID is equals to the Organization name (o) in the subject field of the Certificate and the Part Number Identifier is resulting of the Identifier computation as defined in section 2.3.1.1 from the URN determined by aNID and aNSS.

11.2.10 FAC (Factory) Certificate

There shall be a FAC CA per PN CA and per equipment (HSM) issuing the OFL Certificates.

The presence of this CA in the Certification Path is optional and dependent on a risk management decision related to the volume of CERT.OFL.ECDSA Certificates to issue.

This Certificate is a CA Certificate as defined in [RFC 5280] section 3.2.

The following table describes the FAC (FACtory) Certificate data related to CERT.FAC.ECDSA Certificate.

Table 11-10: FAC ECDSA X.509 Certificate Format

Field	Value Description	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued by the PN CA.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.
	issuer	Distinguished Name of the PN CA that has signed this Certificate.
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Shall be the same as in the issuing certificate CERT.CI$_{OFL}$.ECDSA defined in Table 11-1.

Field	Value Description				
				Reference	Ref Section
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID the PK.PN.ECDSA public key that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the identifier of the public key bound in this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	Digital Signature of the OFL Certificate		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-fac	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.11 IMO Certificate

11.2.11.1 Description

An OFL Authority is an Image Owner getting privilege rights on the OFL, therefore there is no dedicated certificate for the OFL Authority. This Certificate enables the OFL Authority who is an Image Owner and allows the OFL Operation for transfer of rights as defined in section 8.6.

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

11.2.11.2 Format

The following table describes the Image Owner Certificate data related to CERT.IMO.ECDSA.

Table 11-11: IMO ECDSA X.509 Certificate Format

Field	Value Description	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Shall be unique for each Certificate issued by the OFL Authority.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA Certificate.
	issuer	Any
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

Field	**Value Description**				
	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value. Apply rules defined in Table 11-1 related to CERT.CI$_{IDS}$.ECDSA.		
				Reference	**Ref Section**
	Extensions	id-ce-authorityKeyIdentifier	Authority Key Identifier	[RFC 5280]	4.2.1.1
		false			
		keyIdentifier [0]	SKID the public key of the CA that has to be used to verify this Certificate.		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	SKID of this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	OFL Authority transfer of rights		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-imo	See section 11.1.3.2		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.12 ARP Certificate

11.2.12.1 Overview

The following table describes the Certificate data related to CERT.ARP.ECDSA signed by the OFL Authority, or the TRSH Maker. A valid ARP Certificate shall expose an Effective Date earlier than the time stamp date (as defined in section 11.3.4) and not older than the Expiration Date.

The ARP Certificate is verified by using PK.IMO$_{OFLA}$.ECDSA or PK.TRSH$_{M}$.ECDSA as expressed in section 9.3.

This Certificate is an End Entity Certificate as defined in [RFC 5280] section 3.2.

11.2.12.2 Description

The following table describes the content of the CERT.ARP.ECDSA Certificate.

Table 11-12: ARP Certificate

Field	Value Description	
tbsCertificate	Data to be signed	
	Field	**Value Description**
	version	Shall be version 3 (value is 2) as extensions are used in this specification.
	serialNumber	Serial number that shall be unique for each Certificate issued by the OFL Authority or the TRSH Maker.
	signature	Contains the algorithm identifier and parameters used by the issuer to compute the value of the field 'signatureValue'. Apply rules defined in Table 11-1 related to CERT.CI$_{OFL}$.ECDSA Certificate.
	issuer	Distinguished Name of the OFL Authority or the TRSH Maker that has signed this Certificate.
	validity	Validity period of the Certificate.

	subject	Distinguished Name of the Certificate. It shall include at least 'organization' and 'commonName' attributes.
		Example of a DN:
		`c = US`
		`l = New York`
		`o = ACME`
		`cn = ACME IDS`
		`e = ids@acme.com`

	subjectPublicKeyInfo	Contains the algorithm identifier, parameters and public key value. Apply rules defined in Table 11-1 related to CERT.CI$_{OFL}$.ECDSA.

Field	Value Description				
				Reference	Ref Section
	Extensions	id-ofl-arp-operation	Access Rights Pattern	OFL 2.0	11.2.12.3
		true			
		ARP-Operation-Extension	Extension as defined in section 11.2.12.3.2		
		id-ce-subjectKeyIdentifier	Subject Key Identifier	[RFC 5280]	4.2.1.2
		false			
		keyIdentifier [0]	Contains the identifier of the public key bound in this Certificate.		
		id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyCertSign (5)	This Certificate does not sign anything		
		id-ce-certificatePolicies	Certificate Policies	[RFC 5280]	4.2.1.4
		true			
		id-role-arp	See section 11.1.3.2		
		id-ofl-arp-filter-isid	ISID Filter	OFL 2.0	11.2.12.3.3
		false			
		ISID	Filtered by Image Session Identifier (ISID)		
		id-ofl-arp-filter-bnpi-id	BNPI	OFL 2.0	11.2.12.3.4
		false			
		BNPI	Filtering by BNPI		
		id-ofl-arp-filter-group-id	Firmware Group Identifier Filter	OFL 2.0	11.2.12.3.5
		false			
		UUID$_G$	Filtering by Firmware Group Identifier		
		id-ofl-arp-filter-pn	Part Number Identifier Filter	OFL 2.0	11.2.12.3.6
		false			
		PART-NUMBER	Part Number Identifier as defined in section 2.3		
signatureAlgorithm			See section 11.1.3		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3		

11.2.12.3 ARP Certificate Extensions

11.2.12.3.1 Overview

The ARP Certificate extensions may be conditional. A successful ARP validation involves that all ARP Certificate extensions shall be successfully validated.

Each ARP extension is an X509v3 Certificate extension for the ARP Certificate.

The following ASN.1 description defines the object identifiers (OID) of the ARP Certificate extensions.

```
-- ASN1START
id-ofl-arp-extension OBJECT IDENTIFIER ::= {id-globalplatform ofl(10) arp-extension(2)}

id-ofl-arp-operation                  OBJECT IDENTIFIER ::= {id-ofl-arp-extension 1}
id-ofl-arp-filter-isid                OBJECT IDENTIFIER ::= {id-ofl-arp-extension 2}
                                      --Filtered by ISID
id-ofl-arp-filter-bnpi                OBJECT IDENTIFIER ::= {id-ofl-arp-extension 3}
                                      --Filtered by Batch Number
id-ofl-arp-filter-group-id            OBJECT IDENTIFIER ::= {id-ofl-arp-extension 4}
                                      --Filtered by Group Identifier
id-ofl-arp-filter-pn-id               OBJECT IDENTIFIER ::= {id-ofl-arp-extension 5}
                                      --Filtered by Part Number Identifier

-- ASN1STOP
```

11.2.12.3.2 ARP Operation Extension

This extension is mandatory and its content is used in the procedure defined in section 9.3.

The following ASN.1 description defines the ARP Operation extension.

```
-- ASN1START

ARP-Operation-Extension               ::= [APPLICATION 4] SEQUENCE
{
aOP-ARP                               Operation-code,
                                      -- OPARP: ARP on the the Image operation (OPIMA)
aIMOCertificate                      Certificate OPTIONAL
                                      --Image Owner Certificate

}
-- ASN1STOP
```

Where:

aOP-ARP	:	Authorized OFL Operation Codes defined in Table 7-3
aIMOCertificate	:	The Certificate of the Image Owner for the transfer of the OFL Authority

aOP-ARP authorizes the OFL Operation Codes requested in the Image Descriptor called OP_{IMA}. The ARP description and the Operation Codes share the same semantic.

Table 11-13: ARP Descriptions

Right	Description
eUpdateFirmwareAllowed	This right allows an Image Owner to perform an update on an existing Firmware.
eLocalFirmwareDeleteAllowed	This right allows an Image Owner to allow the local delete of a Firmware.
eDeleteFirmwareSession	This right allows an Image Owner to delete a Firmware Session and to erase the associated Firmware.
eDisableFirmware	This right allows an Image Owner to set the Firmware Session to the DISABLED state.
eDisableOFL	This right allows an Image Owner to disable OFL. The OFL switches to the DISABLED state, prohibiting any Firmware from being executed except the OFL.
eEnableFirmware	This right allows an Image Owner to enable the Firmware Session, allowing the corresponding enabled Firmware to be executed.
eEnableOFL	This right allows an Image Owner to switch the OFL to the ENABLED state.
eEraseFirmware	This right allows an Image Owner to erase a Firmware without deleting its Firmware Session.
eLoadFirmware	This right allows an Image Owner to create a Firmware Session.
eTransferRights	This right allows an Image Owner to transfer The OFL Authority role to another Image Owner.
eTerminateOFL	This right allows an Image Owner to set the OFL to the TERMINATED state. This operation is irreversible.
eUnlockOFL	This right allows an Image Owner to set the OFL to the UNLOCKED state.
eUpdatePolicy	This right allows an Image Owner to update Local ARP Policy.

11.2.12.3.3 ISID Filter Extension

This filter allows the validation of an ARP if an ISID associated to a pending Image Session matches the extnValue[1] value containing the ISID value.

When present, the ISID value provided in this ISID tag shall be the same as the ISID provided within ATK.IDS$_2$.ECKA.

- The following ASN.1 description defines the ISID:

```
-- ASN1START
ISID                                    ::= [PRIVATE 18]UUID
                                        --ISID as defined in section 3.2
-- ASN1STOP
```

[1] extnValue is a field of the Certificate extension as defined in [RFC 5280] section 4.1.

11.2.12.3.4 BNPI Filter Extension

This filter allows the validation of an ARP if a BNPI is a pseudonym of the Batch Number (BN) of a Secure Hardware Platform. extnValue shall contain the BNPI as a pseudonym of the Batch Number of a Secure Hardware Platform. The BNPI is defined in section 10.2.

- The following ASN.1 description defines the BNPI:

```
-- ASN1START
BNPI                                       ::= [PRIVATE 19]UUID
                                           -- BNPI is defined in section 10.2.

-- ASN1STOP
```

11.2.12.3.5 Group Identifier Filter Extension

This filter allows the validation of an ARP if the Group Identifier of the Firmware ($UUID_G$) to load matches the extnValue value.

extnValue shall contain the Firmware Group Identifier ($UUID_G$) to match. The Firmware Group Identifier is defined in section 2.3.

- The following ASN.1 description defines the $UUID_G$:

```
-- ASN1START
UUIDG                                      ::= UUID
                                           -- Firmware Group Identifier (UUIDG)
-- ASN1STOP
```

11.2.12.3.6 Part Number Identifier Filter Extension

This filter allows the validation of an ARP if the Part Number Identifier matching the extnValue value. extnValue shall contain the Part Number Identifier (PART_NUMBER) to match. The Part Number Identifier is defined in section 2.3.

- The following ASN.1 description defines the PART-NUMBER:

```
-- ASN1START
PART-NUMBER                                ::= [PRIVATE 20]UUID
                                           --PART_NUMBER: Part Number Identifier
-- ASN1STOP
```

11.2.12.3.7 Organization Dependent Extension

This section is intentionally blank and aims at informing other organizations that they may extend the ARP extensions for specific purposes involving the OFL Authority or the TRSH Maker.

11.3 Authentication Tokens

11.3.1 Authentication Token Verification

11.3.1.1 Common Authentication Token Verification

The Authentication Token verification algorithm is the following:

- The public key algorithm and parameters shall be checked.
- All public key algorithm and parameters for key agreement shall be equal to the public key algorithm and parameters of CERT.IDS$_1$.ECKA.
- The signature on the Authentication Token shall be verified using public key algorithm, the public key, and the public key parameters of the End Entity Certificate of the Certification Path.
- The key usage extension is checked to ensure that is allowed to perform a key agreement.
- Any other critical extensions are recognized and processed.

If this procedure is applied to the Authentication Token with policy violations or any other error condition, then the Authentication Token validation algorithm terminates with an error.

11.3.2 OFL Authentication Token

11.3.2.1 Description

The ATK.OFL.ECKA Token based on a subset of an X.509 Certificate version 3 as follows:

Table 11-14: ATK.OFL.ECKA

Field	Value Description				
tbsToken	Data to be signed and defined as tbsToken$_{OFL}$				
	Field		**Value Description**		
	version		Shall be version 1.		
	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value of ePK.OFL.ECKA.		
	tbsToken		Shall contain ATK-OFL-ECKA-Content		
				Reference	**Ref Section**
	Extensions	id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyAgreement (4)	Key agreement for computing KS1[9] and KS2[9]		
signatureAlgorithm			Shall be the same as the algorithm used to compute the signature in CERT.OFL.ECDSA		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3.1		

The ATK.OFL.ECKA is a Token as defined in section 6.2.1 that contains in its aATK-Content an ATK-OFL-ECKA-Content data object detailed as follows:

```
-- ASN1START
ATK-OFL-ECKA-Content ::= [APPLICATION 5]SEQUENCE
{
aOFLVersion                            OFLVersion ,
                                       --Major (MM) and Minor (mm) version of the OFL
aEncryption-Type                       Encryption-Type,
                                       --Type of Encryption
aISID                                  ISID,
                                       --ISID as Image Session Identifier
aM-TRSH-Ext                            EncryptedBlock
                                       --Encryption of the TRSH-Extensions with KTRSH-Ext
}
TRSH-Extensions                        ::= Extensions
                                       -- X.509v3 Extension: object identifiers for TRSH/IDS
extensions

id-ofl-trsh-parameter                  OBJECT IDENTIFIER ::= {id-globalplatform ofl(10) trsh-
parameter(5)}
id-ofl-trsh-parameter-ofla             OBJECT IDENTIFIER ::= {id-ofl-trsh-parameter ofla (1)}
id-ofl-trsh-parameter-is-pseudo        OBJECT IDENTIFIER ::= {id-ofl-trsh-parameter is-pseudo (2)}
id-ofl-trsh-parameter-shppi            OBJECT IDENTIFIER ::= {id-ofl-trsh-parameter shppi (3)}

-- ASN1STOP
```

Where:

aOFLVersion	:	Major (MM) and Minor (mm) version of the OFL: MMmm BCD Format. The major and minor release version defined in the current document specification is '0200'.
aEncryption-Type	:	Type of encryption: eGCM-AES-128 or eGCM-AES-256.
aISID	:	Identifier of the Image Session (ISID).
aM-TRSH-Ext	:	Result of the encryption of the TRSH-Extensions using $K_{TRSH\text{-}Ext}$ key.

11.3.2.2 Extensions

The list of extensions in TRSH-Extensions shall contain the IDS1-Extensions defined in section 6.2.2.2.

The list of extensions in TRSH-Extensions shall contain the extension described by the following parameters:

extnID	:	id-ofl-trsh-parameter-ofla
critical	:	true
extnValue	:	$\text{UUID}_{\text{OFLA}}$ as the OFL Authority Identifier

```
-- ASN1START
UUIDOFLA                                  ::= UUID
-- ASN1STOP
```

The list of extensions in TRSH-Extensions shall contain the extension described by the following parameters:

extnID	:	id-ofl-trsh-parameter-is-pseudo
critical	:	true
extnValue	:	IsPseudonym as a Boolean that is true (1) if CODE_M is an SHPPI of the TRSH as defined in section 7.3, else false (0)

```
-- ASN1START
IsPseudonym                               ::= [PRIVATE 21]BOOLEAN
                                          -- True (1) if CODE_M is a pseudonym of the TRSH
-- ASN1STOP
```

If CODE_M is equal to an SHPPI, then the list of extensions in TRSH-Extensions shall contain the extension described by the following parameters:

extnID	:	id-ofl-trsh-parameter-shppi
critical	:	false
extnValue	:	A deterministic PI and defined in section 10.2. The SHPPI shall be generated from the URN present in the IDS Credentials Parameter extension (id-ofl-ids-parameter-urn) defined in section 6.2.2.

```
-- ASN1START
SHPPI                                     ::= [PRIVATE 22]OCTET STRING (SIZE(16))
                                          --Deterministic SHPPI
-- ASN1STOP
```

11.3.2.3 Organization Dependent Extensions

This section is intentionally blank and aims at informing other organizations that they may extend the TRSH-Extensions for specific purposes.

11.3.3 IDS Authentication Tokens

11.3.3.1 IDS$_2$ Authentication Token

11.3.3.1.1 Description

The ATK.IDS$_2$.ECKA Authentication Token based on a subset of an X.509 Certificate version 3 as follows.

Table 11-15: ATK.IDS$_2$.ECKA

Field	Value Description				
tbsToken	Data to be signed and defined as tbsToken$_{IDS2}$				
	Field		**Value Description**		
	version		Shall be version 1.		
	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value of ePK.IDS$_2$.ECKA.		
	tbsToken		Shall contain ATK-IDS2-ECKA-Content		
				Reference	**Ref Section**
	Extensions	id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
		true			
		keyAgreement (4)	key agreement for computing KS2[9]		
signatureAlgorithm			Shall be the same as the algorithm used to compute the signature in CERT.IDS$_2$.ECDSA		
signatureValue			Signature computed according to an algorithm listed in section 11.1.3.1		

The ATK.IDS$_2$.ECKA is a Token as defined in section 6.2.1 that contains in its aATK-Content an ATK-IDS$_2$-ECKA-Content data object detailed as follows:

```
-- ASN1START
ATK-IDS2-ECKA-Content                    ::= [APPLICATION 6]SEQUENCE
{
aOFLVersion                              [0]OFLVersion ,
                                         --Major (MM) and Minor (mm) version of the OFL

aISID                                    [1]ISID OPTIONAL,
                                         --ISID as defined in section 2.3.1.2
aIDS-Revocation-Status                   [2]OCTET STRING OPTIONAL,
aEncryption-Type                         [3]Encryption-Type,
                                         --Type of symetrical Encryption used for aMIDS2-Extensions
aMIDS2-Extensions                        [4]EncryptedBlock
                                         --Encryption of the IDS2-Extension with KIDS-Ext
}
IDS2-Extensions                          ::= Extensions

-- ASN1STOP
```

Where:

aOFLVersion	:	Major (MM) and Minor (mm) version of the OFL: MMmm BCD Format. The major and minor release version defined in the current document specification is '0200'.
aISID	:	Identifier of the Image Session (ISID)
aIDS-Revocation-Status	:	List of the OCSP responses as defined in section 11.1.3.4.3.
aEncryption-Type	:	The encryption used for aM-CERT-OFL. Shall be set to eGCM-AES-128 or eGCM-AES-256.
$aM_{IDS2\text{-}Extensions}$	:	Encryption of the IDS2-Extensions with $K_{IDS\text{-}Ext}$. IDS2-Extensions is a generic conveyor for any metadata (use case dependent).

Note: aIDS-Revocation-Status may be moved to the metadata of the protocol tunneling the OFL security protocol, as the aIDS-Revocation-Status information is mainly useful for asymmetric Certification Path verification as defined in section 11.1.2.3.

11.3.3.1.2 Organization Dependent Extensions

This section is intentionally blank and aims at informing other organizations that they may extend the IDS2-Extensions for specific purposes.

11.3.3.2 IDS$_1$ Authentication Token

11.3.3.2.1 Description

The ATK.IDS$_1$.ECKA Authentication Token is based on a subset of an X.509 Certificate version 3 as follows.

Table 11-16: ATK.IDS$_1$.ECKA

Field	**Value Description**				
tbsToken	Data to be signed and defined as $tbsToken_{IDS1}$				
	Field		**Value Description**		
	version		Shall be version 1.		
	signature		Shall contain the same content as the signatureAlgorithm		
	subjectPublicKeyInfo		Contains the algorithm identifier, parameters and public key value of ePK.OFL.ECKA.		
	tbsToken		Shall contain ATK-IDS$_1$-ECKA-Content		
				Reference	**Ref Section**
	Extensions	id-ce-keyUsage	Key Usage	[RFC 5280]	4.2.1.3
	Extensions	true			
	Extensions	keyAgreement (4)	The public key is not used.		

signatureAlgorithm		Shall be the same as the algorithm used to compute the signature inCERT.IDS$_2$.ECDSA
signatureValue		Signature computed according to an algorithm listed in section 11.1.3.1

The ATK.IDS$_1$.ECKA is a Token as defined in section 6.2.1 that contains in its aATK-Content an ATK-IDS$_1$-ECKA-Content data object detailed as follows:

```
-- ASN1START
ATK-IDS1-ECKA-Content                  ::= [APPLICATION 7]SEQUENCE
{
aOFLVersion                            [0]OFLVersion ,
                                       --Major (MM) and Minor (mm) version of the OFL
aISID                                  ISID,
                                       --ISID as Image Session Identifier
aPN-CertificationPath                  CertificationPath,
                                       --FAC Certificates
aM-Time-Stamp                          EncryptedBlock,
                                       -- Encrypted time stamp by the key KTIME-STAMP
aTimeStamp                             TimeStamp,
                                       --plaintext time stamp
aTRSH-Extensions                       Extensions
                                       --TRSH-Extensions


}
-- ASN1STOP
```

Where:

aOFLVersion	:	Copy of the same parameter extracted from ATK-OFL-ECKA-Content ATK-Content
aISID	:	Copy of the same parameter extracted from ATK-OFL-ECKA-Content ATK-Content
aPN-CertificationPath	:	Copy of the same parameter extracted from ATK-OFL-ECKA-Content ATK-Content
aM-Time-Stamp	:	Encrypted time stamp by using the key $K_{TIME\text{-}STAMP}$
aTimeStamp	:	Plaintext time stamp provided by HSM_IDS$_1$ as defined in section 11.3.4
aTRSH-Extensions	:	List of the TRSH-Extensions decrypted from $aM_{TRSH\text{-}Ext}$ extracted from ATK-OFL-ECKA-Content ATK-Content as defined in section 11.3.2.1

11.3.3.2.2 Organization Dependent Extensions

This section is intentionally blank and aims at informing other organizations that they may extend the TRSH-Extensions for specific purposes.

11.3.4 Time Stamp

The time stamp contains the date and time from a secure clock synchronized with a Time Stamp Authority (TSA) as defined in [ANSI X9.95]. The following ASN.1 description illustrates the time stamp.

```
-- ASN1START
TimeStamp                             ::= [PRIVATE 23]DATE-TIME
-- ASN1STOP
```

Annex A ASN.1 END

```
-- ASN1START
END
-- ASN1STOP
```

Annex B eGCM based on AES-128 and AES-256 – Description (Normative)

This annex describes an authenticated encryption mechanism, eGCM, that is a variant of the well-known Galois/Counter Mode (GCM) mechanism. This annex describes the use of both 128-bit and 256-bit keys.

B.1 Introduction

When selecting an authenticated encryption mechanism for an Open Firmware Loader in a TRSH, the main requirements identified were the following:

- High performance: Enables the two mechanisms (Decryption and Tag Verification) to execute in parallel when it is possible to use in parallel a hardware accelerator “AES” and a hardware accelerator “multiplier”. Also supports the case where only a hardware accelerator AES is available.
- Protection against side-channel attacks: Limits the needs of counter-measures against side-channel analysis via the use of some specificities in the design of the selected mechanisms.

These requirements apply only for the decryption phase since the encryption phase is executed in a secure environment.

The GCM mechanism [INDO 2004] was pre-selected as it fulfills the first requirement. Galois/Counter Mode (GCM) is a block-cipher mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. It can be implemented in hardware to achieve high speed operations with low cost and low latency. Software computation can reach a high level of performance by using table-driven field operations. It uses mechanisms that are supported by a well understood theoretical foundation, and its security follows from a single reasonable assumption about the security of the block-cipher. Moreover, there is a security proof on GCM [CRYPTO 2012], [INDO 2004].

However, GCM does not appear to be the best choice for the second requirement. This is why several modifications were considered to improve the protection ***“by design”*** and reduce the need for additional countermeasures against side-channel analysis.

Thus, eGCM is a variant of the authenticated encryption mode GCM [INDO 2004] with a few modifications.

The sections below highlight the main differences between GCM and eGCM implementations.

B.1.1 Operating Mode

In the GCM specification, the counter mode appeared to be the best choice for high-speed encryption; it admits pipelined and parallelized implementations with a minimal computational latency mandatory when high data rates are involved.

However, for an implementation that is limiting the use of counter-measures against side-channel analysis for the block cipher, the counter mode is not the best choice. In that case, an attacker would have full knowledge of the evolution of the counter value, allowing a DPA attack against the block cipher to recover its secret key. Even if the counter is kept secret, by design, it could be possible to observe the differences between the successive counter values.

By contrast, the operating mode chosen for eGCM is the Output FeedBack mode (OFB). Using the OFB mode, the input data of the block-cipher is better protected against DPA attacks, an attacker no longer being able to access the evolution of the counter values. More precisely, the cryptanalyst will need to know the plaintexts corresponding to a set of ciphertext values. Obtaining these plaintext values, introduced into the block-cipher, is not an easy task in practice. During the OFB process, a secret value is used as initialization vector and the ciphertext blocks are exclusive-ored with the output blocks of the block-cipher. In that way, the need of countermeasures to protect the block-cipher against side-channel analysis is reduced compared to the use of the classical counter mode. It is nevertheless assumed that the attacker does not have access to plaintext values linked with the ciphertext values.

B.1.2 Initialization Step

The GCM mode has an initialization vector IV that can have any number of bits between 1 and 2^{64}, while the eGCM mode has two secret initialization vectors, EIV and IV, each 128 bits long.

The value EIV is used to compute the starting value of the OFB encryption mode whereas the value IV is used as a prefix of the plaintext message for the computation of the authentication tag. The idea behind the use of IV, which is assumed to be a secret value, is to protect the value H by masking it, prior to the mixing of H with the plaintext block values. This modification has been done to avoid straightforward DPA attacks against the H value.

B.1.3 Tag Computation on Plaintext Data

The tag is computed on a plaintext data instead of on a ciphertext data. The main reason is that, in the process, the data is decrypted, then written in the memory, and finally read from the memory for the tag value computation. Therefore, the authenticity of the data that is verified is the authenticity of the data read from memory. If an error occurs during the decryption phase, the writing phase, and/or the reading phase, it will therefore be detected at the tag verification phase.

B.1.4 Regular Updates of the Key

One key value is used for only one single ciphertext encryption whose length is limited.

B.2 Preliminaries

We follow the notation described in [CRYPTO 2012] with some changes.

Let $\{0,1\}^*$ be the set of all bit strings, and for an integer $\ell \geq 0$, let $\{0,1\}^\ell$ be a set of ℓ-bit strings. For a bit string $X \in \{0,1\}^*$, $|X|$ is the length in bits and $|X|_\ell = \lceil |X|/\ell \rceil$ is the length in ℓ-bit blocks. The empty string is denoted by ε. Let 0^ℓ and 1^ℓ denote the bit strings of ℓ zeroes and ones, respectively. For X, $Y \in \{0,1\}^*$, we write $X||Y$ or (X,Y) their concatenation. For a bit string X whose length in bits is a multiple of ℓ, we write its partition into ℓ-bit strings as $(X[1],...,X[x]) \xleftarrow{\ell} X$, where $X[1],...,X[x] \in \{0,1\}^\ell$ are unique bit strings such that $X[1]||...||X[x] = X$. For a finite set χ, $\#\chi$ denotes its cardinality, and $X \xleftarrow{\$}$ means the uniform sampling of an element from χ and assigning it to X.

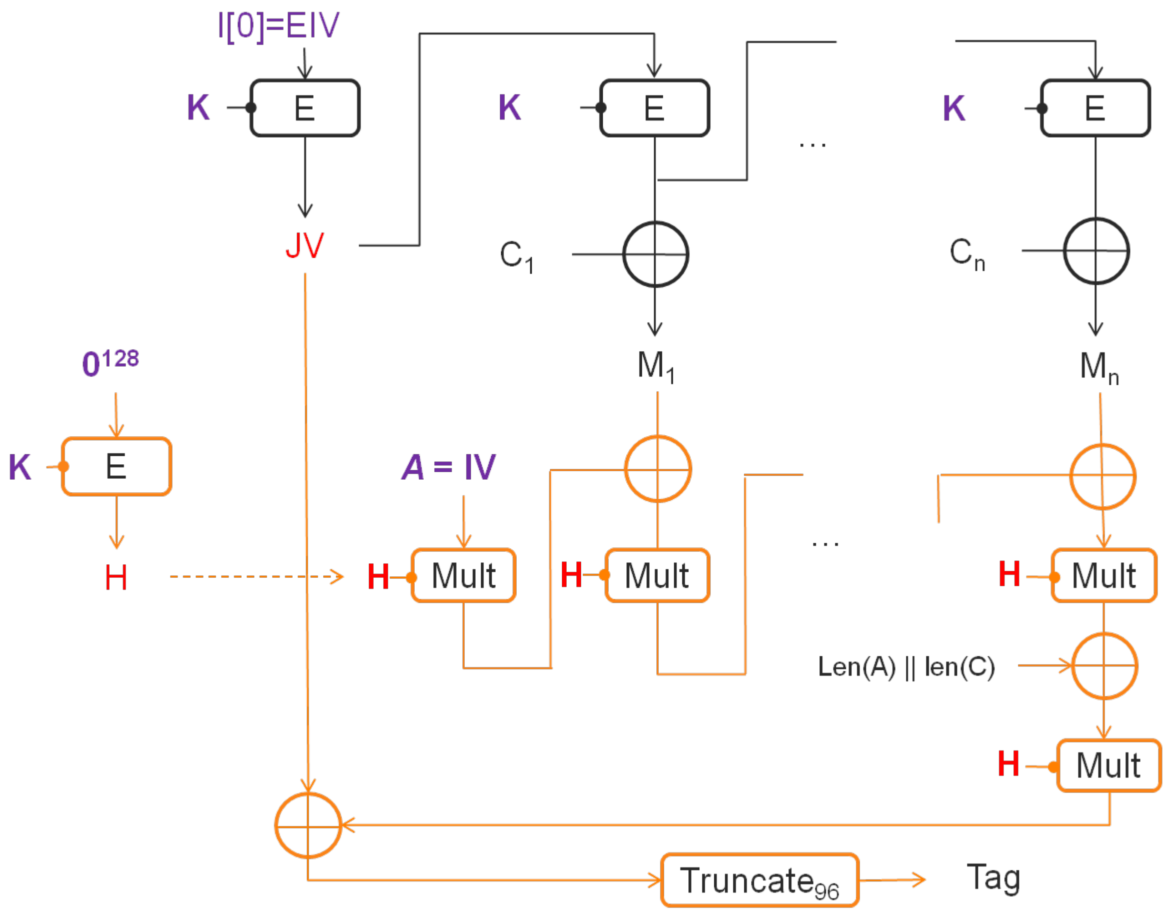
The eGCM mechanism is built upon a block cipher E: $\mathrm{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ and a multiplication over the field $GF(2^{128})$. The block cipher used is AES, then $n = 128$. We write E_K for the permutation specified by $K \in \mathrm{K}$, and $C = E_K(M)$ for the ciphertext of a plaintext $M \in \{0,1\}^n$ under the key $K \in \mathrm{K}$. For $i \geq 1$, $E^i(X)$ means that we apply E on X for i times. We let $E^0(X) = X$ and $E^{-i}(X)$ the inverse function of $E^i(X)$.

The set of n-bit strings, $\{0,1\}^n$, is also regarded as $GF(2^n)$, the finite field with 2^n elements. An n-bit string $a_{n-1} \ldots a_1 a_0 \in \{0,1\}^n$ corresponds to a formal polynomial $a(x) = a_{n-1} + a_{n-2}x + \_\,\_\,\_ + a_1 x^{n-2} + a_0 x^{n-1} \in GF(2)[x]$. When $n = 128$, the irreducible polynomial used in eGCM is $p(x) = 1 + x + x^2 + x^7 + x^{128}$.

B.3 Specification of eGCM

eGCM is parametrized by a block cipher E: $\mathrm{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ and a tag length τ. eGCM[E,τ] describes eGCM that uses E and τ as parameters. Let eGCM-$\mathcal{E}$ be the encryption algorithm defined in section B.3.4 and eGCM-$\mathcal{D}$ be the decryption algorithm defined in section B.3.5. eGCM uses the OFB mode defined in section B.3.2 and the polynomial hash function over $GF(2^n)$, as defined in section B.3.3. Figure B-1 illustrates the overall structure of eGCM-$\mathcal{D}$.

Figure B-1: eGCM Decryption Phase



B.3.1 Inputs and Outputs for Decryption

The authenticated decryption operation has the following inputs:

- A secret key K of length n with n = 128 bits for AES-128 and n = 256 bits for AES-256
- A first secret initialization vector EIV of length n with n = 128 bits such that $EIV \neq 0^{128}$
- A second secret initialization vector IV of length n with n = 128 bits
- A ciphertext C, which can have any number of bits between 0 and 2^{32}

The operation has the following outputs:

- A plaintext value P whose length is exactly that of the ciphertext C
- An authentication tag T whose length is 96 for AES-128 and 128 for AES-256. The tag length is denoted by τ.

B.3.2 OFB Mode for Encryption/Decryption

The update function of the nonce EIV is built upon the block cipher E using the OFB mode. When using eGCM, the starting value for the OFB mode is initialized as follows: $I[0] = EIV$, where EIV is a secret value. Then, the values $I[1],\ldots,I[m]$ are computed as described in Algorithm 1 below, where m is the number of blocks of the plaintext/ciphertext message. For the use of OFB in encryption mode, X is the plaintext message and Y is the ciphertext message, and for the use of OFB in decryption mode, X is the ciphertext message and Y is the plaintext message.

```
Algorithm 1    OFB$_K(I[0],X)$
  $m \leftarrow |X|_n$
  for $j \leftarrow 1$ to $m$ do
    $I[j] \leftarrow E_K(I[j-1])$
  end for
  $Y \leftarrow X \oplus I$
  return $Y$
end
```

B.3.3 Function eGHASH for Tag Computation

The tag value is computed on the plaintext message using the algorithm eGHASH where $H = E_K(0)$.

```
Algorithm 2    eGHASH($IV,M$)
    $\mu \leftarrow n|M|_n - |M|$
    $X \leftarrow IV \parallel M \parallel 0^{\mu} \parallel Len(IV)||Len(M)$
    $(X[1],...X[m]) \leftarrow X$
    $Y \leftarrow 0^n$
    for $j \leftarrow 1$ to $m$ do
    $Y \leftarrow H \cdot (Y \oplus X[j])$
    end for
    return $Y$
end
```

B.3.4 Encryption Algorithm

The algorithm eGCM-$\mathcal{E}$ takes a key $K \in \mathrm{K}$, a nonce $EIV \in \{0,1\}^n$, a second nonce $IV \in \{0,1\}^n$ and a plaintext $M \in \{0,1\}^*$ as inputs and returns a pair of a ciphertext $C \in \{0,1\}^*$ and a tag $T \in \{0,1\}^{\tau}$ as an output.

```
Algorithm 3    eGCM− $\mathcal{E}_K^{EIV,\,IV}(M)$
    $H \leftarrow E_K(0^n)$
    $I[0] \leftarrow EIV$
    $I[1] \leftarrow E_K(I[0])$
    $C \leftarrow OFB_K(I[1],M)$
    $\check{T} \leftarrow I[1] \oplus eGHASH_H(IV,M)$ $T \leftarrow msb_{\tau}(\check{T})$
    return $(C,T)$
end
```

B.3.5 Decryption Algorithm

The algorithm eGCM-$\mathcal{D}$ takes as inputs a key $K \in \mathrm{K}$, nonces $EIV \in \{0,1\}^n$ and $IV \in \{0,1\}^n$, and a pair of ciphertext, tag ($C \in \{0,1\}^*, T \in \{0,1\}^\tau$) and returns either a plaintext $M \in \{0,1\}^*$ or the symbol ⊥ indicating that the inputs are invalid.

```
Algorithm 4        eGCM−$\mathcal{D}_K^{EIV,\, IV}(C,T)$
   $H \leftarrow E_K(0^n)$
   $I[0] \leftarrow EIV$
   $I[1] \leftarrow E_K(I[0])$
   $M \leftarrow \mathrm{OFB}_K(I[1],C)$
   $\check{T}^* \leftarrow I[1] \oplus \mathrm{eGHASH}(IV,M)$
   $T^* \leftarrow msb_\tau(\check{T}^*)$
   if $T \neq T^*$ then
           return $\perp$
   else
           return $M$
   end if
end
```

Annex C VPP Firmware (Normative)

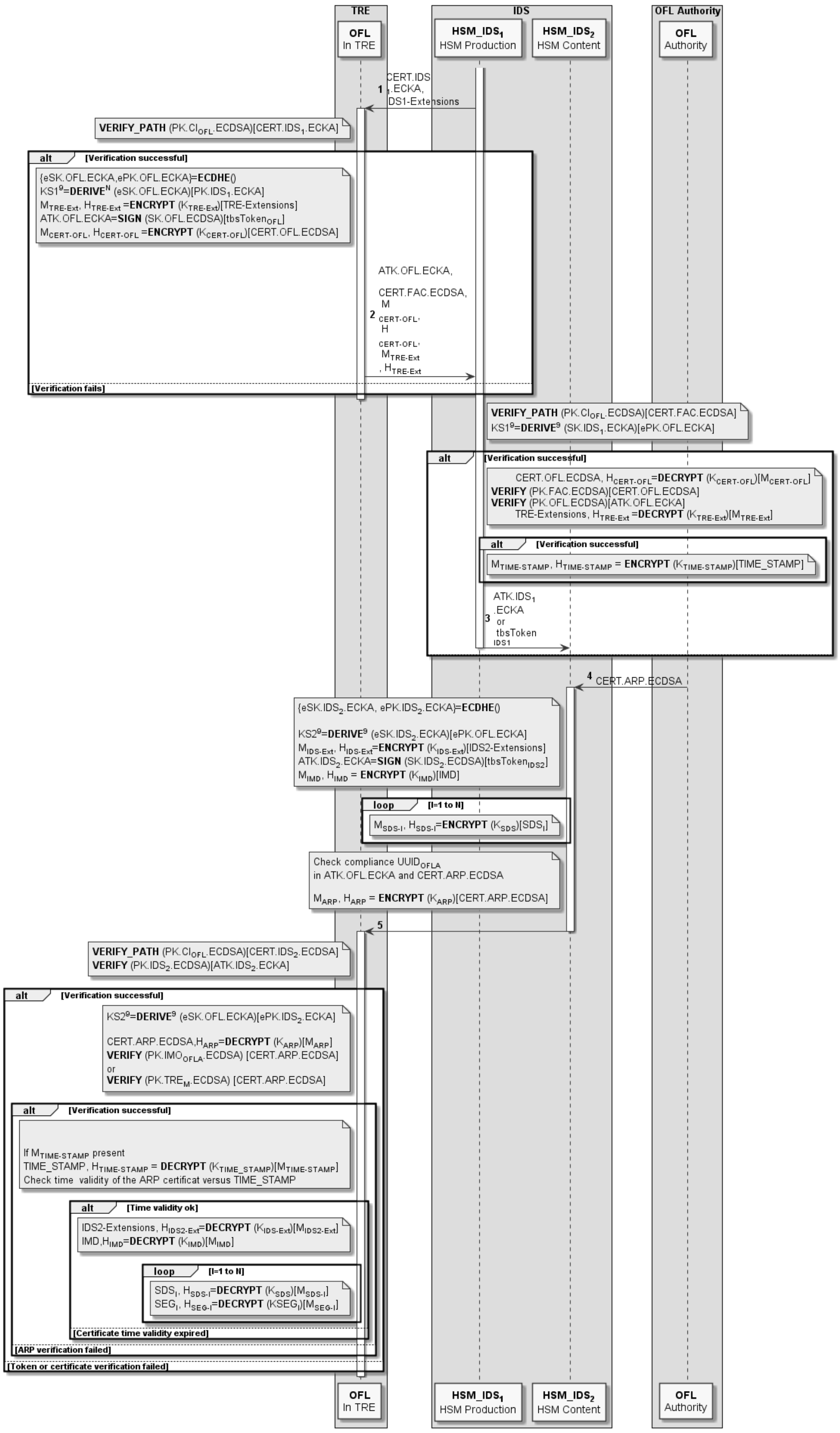
This annex describes the rules that shall be used by the Firmware Maker to provide a VPP Firmware consistent with the OFL loading procedure.

The Firmware is encapsulated and protected within an OFL Image prior to loading it into a TRSH. An OFL Image encapsulates a collection of OFL segments as defined in section 7.1. The first OFL segment shall contain only the whole VPP Firmware Header as defined in GlobalPlatform VPP Firmware Format [VFF] section 5.

OFL is responsible to fully write each segment of each Sub-Memory Partition related to a given part of the Firmware. If data extracted from an OFL segment is less than indicated in the header, OFL shall pad the segment by writing zeros ('00').

Annex D Overview of the Security Protocol

Figure D-1: Overview of the Security Protocol



TRE
IDS
OFL Authority
OFL In TRE
HSM_IDS$_1$ HSM Production
HSM_IDS$_2$ HSM Content
OFL Authority
1 CERT.IDS$_1$.ECKA, IDS1-Extensions
VERIFY_PATH (PK.CI$_{OFL}$.ECDSA)[CERT.IDS$_1$.ECKA]
alt [Verification successful]
(eSK.OFL.ECKA,ePK.OFL.ECKA)=ECDHE()
KS1^9=DERIVE^N (eSK.OFL.ECKA)[PK.IDS$_1$.ECKA]
M$_{TRE-Ext}$, H$_{TRE-Ext}$ =ENCRYPT (K$_{TRE-Ext}$)[TRE-Extensions]
ATK.OFL.ECKA=SIGN (SK.OFL.ECDSA)[tbsToken$_{OFL}$]
M$_{CERT-OFL}$, H$_{CERT-OFL}$ =ENCRYPT (K$_{CERT-OFL}$)[CERT.OFL.ECDSA]
2 ATK.OFL.ECKA, CERT.FAC.ECDSA, M$_{CERT-OFL}$, H$_{CERT-OFL}$, M$_{TRE-Ext}$, H$_{TRE-Ext}$
[Verification fails]
VERIFY_PATH (PK.CI$_{OFL}$.ECDSA)[CERT.FAC.ECDSA]
KS1^9=DERIVE^9 (SK.IDS$_1$.ECKA)[ePK.OFL.ECKA]
alt [Verification successful]
CERT.OFL.ECDSA, H$_{CERT-OFL}$=DECRYPT (K$_{CERT-OFL}$)[M$_{CERT-OFL}$]
VERIFY (PK.FAC.ECDSA)[CERT.OFL.ECDSA]
VERIFY (PK.OFL.ECDSA)[ATK.OFL.ECKA]
TRE-Extensions, H$_{TRE-Ext}$ =DECRYPT (K$_{TRE-Ext}$)[M$_{TRE-Ext}$]
alt [Verification successful]
M$_{TIME-STAMP}$, H$_{TIME-STAMP}$ = ENCRYPT (K$_{TIME-STAMP}$)[TIME_STAMP]
3 ATK.IDS$_1$.ECKA or tbsToken$_{IDS1}$
4 CERT.ARP.ECDSA
(eSK.IDS$_2$.ECKA, ePK.IDS$_2$.ECKA)=ECDHE()
KS2^9=DERIVE^9 (eSK.IDS$_2$.ECKA)[ePK.OFL.ECKA]
M$_{IDS-Ext}$, H$_{IDS-Ext}$=ENCRYPT (K$_{IDS-Ext}$)[IDS2-Extensions]
ATK.IDS$_2$.ECKA=SIGN (SK.IDS$_2$.ECDSA)[tbsToken$_{IDS2}$]
M$_{IMD}$, H$_{IMD}$ = ENCRYPT (K$_{IMD}$)[IMD]
loop [I=1 to N]
M$_{SDS-I}$, H$_{SDS-I}$=ENCRYPT (K$_{SDS}$)[SDS$_I$]
Check compliance UUID$_{OFLA}$ in ATK.OFL.ECKA and CERT.ARP.ECDSA
M$_{ARP}$, H$_{ARP}$ = ENCRYPT (K$_{ARP}$)[CERT.ARP.ECDSA]
5
VERIFY_PATH (PK.CI$_{OFL}$.ECDSA)[CERT.IDS$_2$.ECDSA]
VERIFY (PK.IDS$_2$.ECDSA)[ATK.IDS$_2$.ECKA]
alt [Verification successful]
KS2^9=DERIVE^9 (eSK.OFL.ECKA)[ePK.IDS$_2$.ECKA]
CERT.ARP.ECDSA,H$_{ARP}$=DECRYPT (K$_{ARP}$)[M$_{ARP}$]
VERIFY (PK.IMO$_{OFLA}$.ECDSA) [CERT.ARP.ECDSA]
or
VERIFY (PK.TRE$_M$.ECDSA) [CERT.ARP.ECDSA]
alt [Verification successful]
If M$_{TIME-STAMP}$ present
TIME_STAMP, H$_{TIME-STAMP}$ = DECRYPT (K$_{TIME\_STAMP}$)[M$_{TIME-STAMP}$]
Check time validity of the ARP certificat versus TIME_STAMP
alt [Time validity ok]
IDS2-Extensions, H$_{IDS2-Ext}$=DECRYPT (K$_{IDS-Ext}$)[M$_{IDS2-Ext}$]
IMD,H$_{IMD}$=DECRYPT (K$_{IMD}$)[M$_{IMD}$]
loop [I=1 to N]
SDS$_I$, H$_{SDS-I}$=DECRYPT (K$_{SDS}$)[M$_{SDS-I}$]
SEG$_I$, H$_{SEG-I}$=DECRYPT (KSEG$_I$)[M$_{SEG-I}$]
[Certificate time validity expired]
[ARP verification failed]
[Token or certificate verification failed]

Annex E Algorithm to Generate the Mask of Operation Codes (Informative)

The algorithm to generate the Mask of Operation Codes is detailed below in the form of a Word macro (Visual Basic language).

```
Sub GenerateMaskOperationCodes()


Dim MaxY As Integer
Dim MaxX As Integer

Dim aTable(16, 11) As Variant
Dim aFields(16) As String
Dim aMasks(9) As String

ActiveDocument.Bookmarks("TableOperation").Select
MaxX = UBound(aTable, 1)
MaxY = UBound(aTable, 2) + 1

aMasks(1) = "eARPINIT-OFLA"
aMasks(2) = "eARPINIT-TRSH-MAKER"
aMasks(3) = "eARPINIT-DEFAULT"
aMasks(4) = "eMASKARP-STATE-BLANK"
aMasks(5) = "eMASKARP-STATE-LOCKED"
aMasks(6) = "eMASKARP-STATE-UNLOCKED"
aMasks(7) = "eMASKOFL-STATE-TERMINATED"
aMasks(8) = "eMASKOFL-STATE-ENABLE"
aMasks(9) = "eMASKOFL-STATE-DISABLE"


With Selection.Range
    For X = 0 To .Cells.Count - 1
    aTable(Int(X / MaxY), X Mod MaxY) = .Cells(X + 1).Range.Text
    Next X
End With

aLF = Chr(13) + Chr(10)
'ARP masks
S = ""
B = ""
For M = 1 To 9
maskName = Trim(aTable(0, M))
maskName = Left(maskName, Len(maskName) - 2)
maskName = aMasks(M)
S = S + maskName + " Operation-code::={" + aLF
B = B + maskName + " Operation-code::= '"
    Bitstring = ""
    BitstringC = ""
    For L = 1 To 16
        fieldname = RTrim(aTable(L, 0))
        fieldname = Left(fieldname, Len(fieldname) - 2)
        fieldname = Right(fieldname, Len(fieldname) - 1)
        If Len(aTable(L, M)) > 2 Then
        S = S + "e" + fieldname + "," + aLF
        Bitstring = Bitstring + "1"
        BitstringC = BitstringC + "0"
        Else
        S = S + "eC" + fieldname + "," + aLF
        BitstringC = BitstringC + "1"
        Bitstring = Bitstring + "0"
        End If
   Next L
   B = B + Bitstring + BitstringC + "'B" + aLF
   S = Left(S, Len(S) - 3) + aLF + "}" + aLF
Next M

Debug.Print B
Debug.Print S
End Sub
```