GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology
VPP - Concepts and Interfaces
Version 1.0

Public Release

March 2018

Document Reference: GPC_FST_142

Contents

Figures

Tables

1 Introduction

1.1 Audience

This document specifies the concepts, rules, requirements, interfaces related to a Virtual Primary Platform, as outlined in [IUICC Req] and aims at easing the portability of Firmwares designed for secure and Tamper Resistant Elements.

Note: Portability is defined as capability of a program to be executed on various types of data processing systems without converting the program to a different language and with little or no modification [2382].

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://www.globalplatform.org/specificationsipdisclaimers.asp. 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

Table 1-1: Normative References

Standard / Specification	Description	Ref
AIS20	Functionality classes and evaluation methodology for deterministic random number generators, Reference: AIS20, version 1, 02/12/1999, BSI	[AIS20]
AIS31	Functionality classes and evaluation methodology for physical random number generators, Reference: AIS31 version 1, 25/09/2001, BSI	[AIS31]
BSI-CC-PP-0084-2014	Security IC Platform BSI Protection Profile 2014 with Augmentation Packages.	[PP-0084]
FIPS PUB 180-4	Secure Hash Standard (SHS)	[FIPS 180-4]
FIPS PUB 186-4	Digital Signature Standard (DSS)	[FIPS 186-4]
FIPS PUB 197	FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION PUB 197 - Advanced Encryption Standard (AES)	[FIPS 197]
FIPS PUB 198-1	FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION The Keyed-Hash Message Authentication Code (HMAC)	[FIPS 198-1]
FIPS PUB 202	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions	[FIPS 202]
GP OFL	GlobalPlatform Card Technology Open Firmware Loader for Tamper Resistant Element Version 1.3	[OFL]

Standard / Specification	Description	Ref
GP OFL Interface	GlobalPlatform VPP-Network Protocol Extension for the Open Firmware Loader	[EOFL]
GP VPP-Firmware Format	GlobalPlatform VPP-Firmware Format specification	[VFF]
GP VPP-Network Protocol	GlobalPlatform VPP-Network Protocol specification	[VNP]
IETF RFC 2119	Key words for use in RFCs to Indicate Requirement Levels	[RFC 2119]
ISO/IEC 2382:2015	Information technology - Vocabulary	[2382]
Joint Interpretation Library	Joint Interpretation Library: "Application of Attack Potential to Smartcards, v2.9, Jan 2013."	[JIL]
NIST Special Publication 800-38A	Recommendation for Block Cipher Modes of Operation	[NIST SP800-38A]
NIST Special Publication 800-38D	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC	[NIST SP800-38D]
NIST Special Publication 800-57 Part 1 Revision 4	Recommendation for Key Management Part 1: General	[NIST SP 800-57 Pt1 R4]
Remote BIST	Elena Dubrova, Mats Näslund, Gunnar Carlsson, John Fornehed, Ben Smeets. *Two Countermeasures Against Hardware Trojans Exploiting Non-Zero Aliasing Probability of BIST.* DOI 10.1007/s11265-016-1127-4. Springer. J Sign Process Syst. 2016	[BIST]
RFC 2104	HMAC: Keyed-Hashing for Message Authentication	[RFC 2104]
RFC 4122	A Universally Unique IDentifier (UUID) URN Namespace	[RFC 4122]
RFC 4231	Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512	[RFC 4231]
RFC 4492	Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)	[RFC 4492]
RFC 4493	The AES-CMAC Algorithm	[RFC 4493]

Table 1-2: Informative References

Standard / Specification	Description	Ref
ETSI TS 102 221	Smart Cards; UICC-Terminal interface; Physical and logical characteristics	[102 221]
ETSI TS 102 223	Smart Cards; Card Application Toolkit (CAT)	[102 223]
ETSI TS 102 622	Smart Cards; UICC - Contactless Front-end (CLF) Interface; Host Controller Interface (HCI (Release 13)	[102 622]
ETSI TS 103 383	Smart Cards; Embedded UICC; Requirements Specification	[103 383]
ETSI TS 103 384	Smart Cards; Embedded UICC; Technical Specification	[103 384]
GPC_SPE_014	GlobalPlatform Card Specification v.2.2 Amendment D: Secure Channel Protocol '03' v1.1.1.	[HLOS14]

Standard / Specification	Description	Ref
GPC_SPE_025	GlobalPlatform Card Specification v.2.3 Amendment C: Contactless Services v1.2	[HLOS25]
GPC_SPE_034	GlobalPlatform Card Specification v.2.3	[HLOS34]
GPC_SPE_042	GlobalPlatform Card Specification v.2.3 Amendment E: Security Upgrade for Card Content Management v1.1	[HLOS42]
GPC_SPE_093	GlobalPlatform Card Specification v.2.3 Amendment F: Secure Channel Protocol '11'	[HLOS93]
GSMA iUICC PoC PP	GSMA iUICC PoC Group Primary Platform Requirements	[IUICC Req]
ISO/IEC 7816-4	Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange.	[7816-4]
SGP.02	GSMA Remote Provisioning of Embedded UICC Technical specification V3.0	[HLOS02]

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
ABI	Application Binary Interface. The interface between two Programs and using the processor instruction set (with details like registers, stack organization, memory access types, CPU modes...), the sizes, layout, and alignment of basic data types the CPU can directly access. The use of an ABI targets specifically the interface between a Process and the LLOS running in different CPU Modes.
Abstraction	An interface layer that masks underlying implementation differences
Access Group	A logical grouping of software and/or hardware functions to indicate their level of access to other Access Groups.
Accessor	(1) A Program capable of reading the data of a hardware function. (2) A hardware function capable of reading the data of another hardware function.
Address Space	The set of addresses that can be used by a particular Program or functional unit.

Term	Definition
API	Application Programming Interface. The interface between an application of a Program and the latter Program. API use may vary depending on the type of programming language involved.
Bootstrap Program	A Program used for loading the Primary Platform software from a Remote Memory and to instantiate this software. A short Program that is permanently resident or easily loaded into a computer and whose execution brings a larger Program, such as an operating system or its loader, into memory [2382].
COM Process	A Process of VPP Execution Domain providing VPP Application communication as defined in [VNP].
Composite Identifier	Identifier having two parts: Execution Domain type and an enumerated identifier.
Exception	A notification from the kernel to a parent Process about a special condition that occurred in one of its child Processes.
Execution Domain	Property defining the membership of a Process in a logical group of Processes.
Firmware	The data needed to instantiate a VPP Application. See [OFL].
Firmware Format	The data structure of the Firmware. See [VFF].
Firmware Identifier	As defined in [VFF]
Firmware Loader	Program in charge to load or update a Firmware into the TRE. The Firmware Loader may expose extended capability for loading or updating the Primary Platform software.
Forward Compliance	Forward Compliance is the capability to support future needs (e.g. larger Firmware) within the limits of the resources outside the TRE.
HLOS	High Level Operating System. An additional Program encompassed in a VPP Application, that provides further abstraction of resources and services to its HLOS Application(s).
HLOS Application	A Program or interpretable code using the HLOS to deliver a specific use case.
Instance	Concrete occurrence of any object at run time. The creation of an instance from the description of an object (e.g. Firmware, software) is called instantiation.
Integrated TRE	A TRE which is integrated into, and part of, a larger SoC.
IPC	Inter Process Communication based on a shared virtual memory between two Processes.
Kernel Object	Implementation dependent data structure within the kernel that represents a Process, a Mailbox, an IPC or a VRE.
Kernel Object Handle	A runtime representation of an instantiated Kernel Object instantiated by the kernel for a Process.

Term	Definition
Kernel Object Identifier	A unique identifier that allows VPP Application to get a Kernel Object Handle. The identifier is known at development time. Some identifiers are pre-defined as part of VPP.
Key Protection Function	Hardware mechanism to separate key material from CPU access.
LIB Descriptor	Shared library metadata as defined in [VFF].
LLOS	Low level Operating System. A hardware-dependent software running in Privileged CPU Mode and including a microkernel, critical low-level resources and security functions support.
Mailbox	A component owned by a Process capable of receiving Signals from another Process or the kernel.
Main Process	The first Process to run of a VPP Application.
Master Port	The access point to direct memory transfer hardware function.
Memory Partition	Concatenated sub-Memory Partitions as defined in [VFF].
MGT Process	A Process within the VPP Execution Domain responsible for managing VPP Applications.
Microkernel	Minimal software supporting only the functionalities which are not able to run in unprivileged mode such as memory management including their access protection settings, multi-processing support. For simplicity the rest of the present document uses “kernel” as an alias of microkernel.
MLOI	Minimum Level Of Interoperability. The set of minimum system capabilities that are guaranteed to exist in every Primary Platform.
MMF	Memory Management Function as a function in charge to translate a physical address to a virtual address and to apply rules of access (e.g. read, write, execute) according to a Security Perimeter and the nature of the virtual address sub space.
Mutator	(1) A Program capable of writing, modifying or controlling the data of a hardware function. (2) A hardware function capable of writing, modifying or controlling the data of another hardware function.
Non-Shareable Memory Space	Memory space that shall be declared by and accessed by a single Program.
Operating System	Program that controls the execution of other programs and that may provide services such as resource allocation, scheduling, input-output control, and data management [2382].
OTP	One-Time Programmable physical memory cells, such as eFuses that can be integrated in a SoC. These elements are programmed once and retain their programmed value afterwards indefinitely.
Physical Address	The actual, non-translated, addresses that can be used by a particular Program or functional unit.
Physical Address Space	The set of physical addresses that can be used by a particular Program or functional unit as defined in [2382].

Term	Definition
Preemption	The act of temporarily interrupting a Process being managed by a kernel, without requiring its cooperation, and with the intention of resuming the Process at a later time.
Primary Platform	The hardware platform along with a low-level Operating System managing the exceptions, the hardware platform resources and their accesses, the services offering an abstraction of the resources.
Primary Platform Maker	The entity developing and providing the Primary Platform.
Primary Platform Software	Software package that contains the Program and data of the Primary Platform (i.e., LLOS and VPP Processes).
Privileged CPU Mode	CPU mode when dealing with hardware exceptions or when executing privileged instructions.
Process	Independent sequences of execution in CPU unprivileged mode and running within an independent Virtual Address Space. A process may have shared memory with other Processes (e.g. IPC). A Process is scheduled by the LLOS.
Process Descriptor	Process metadata, as defined in [VFF].
Program	Independent set of instructions executed by a CPU.
Remote Audit Function	A mechanism that enables a VPP Application to challenge the authenticity of the Primary Platform.
Remote Memory	Memory located outside the TRE, which can be RAM and NVM
Secure CPU	A CPU supporting security functions for preventing hardware attacks (e.g. DPA, DFA, Side channels).
Security Perimeter	denotes the perimeter of a function on which rules, Access Groups, properties and requirements apply
Service	Process offering an abstraction of a functionality (e.g. communication) providing an interface to another Process.
Shared Memory Space	Memory space that may be accessed by at least two Programs that declare sharing of a specific memory space.
Signal	(1). noun. A fixed bit of information representing a fixed event. (2). verb. The act of sending the signal bit between two Processes or between a Process and the kernel.
System Tick	A fixed duration periodical event that interrupts the Microkernel and is used in Process scheduling.
System VPP Application	A VPP Application having special privileges that enables it to manage other applications. E.g. OFL [OFL].
Thread	A Process within another Process that uses the resources of the latter Process as defined in [2382].
TRE	Tamper Resistant Element as defined in [IUICC Req].
Unprivileged CPU Mode	CPU mode when NOT able to access privileged instructions or when NOT dealing with hardware exceptions.

Term	Definition
Virtual	Pertains to a functional unit that appears to be real, but whose functions are accomplished by other means as defined in [2382].
Virtual Address Space	Set of virtual addresses that can be used by a particular Program or functional unit as defined in [2382].
Virtual Address Space Region	A fixed-size partition of the Virtual Address Space dedicated to a given memory type of a Process, e.g. code, stack, constants, etc.
Virtual Hardware Platform	The virtualized hardware as it appears to a VPP Application
Virtualization	Refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.
VPP	Instance of the Primary Platform at run time that exposes as a virtual Primary Platform (VPP) for a VPP Application. The VPP includes interfaces to services and the kernel and supports the virtualization of resources.
VPP Application	An instance of a Firmware on a VPP; use case dependent and can run an HLOS and its application(s).
VPP Process	A Process within the VPP Execution Domain.
VRE	Virtual REgister is a set of virtual address allowing the access to a hardware function.

1.5 Abbreviations and Notations

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

Table 1-4: Abbreviations

Abbreviation	Meaning
BIST	Built-In Self Test
CPU	Central Processing Unit
CUT	Chip-Under-Test
DFA	Differential Fault attacks defined in [JIL]
DPA	Differential Power Analysis defined in [JIL]
LIB	Library
MGT	Management
MISR	Multiple Inputs Signature Register
NA	Not Applicable
NVM	Non-Volatile Memory
PFS	Perfect Forward Secrecy

Abbreviation	Meaning
PRPG	Pseudo Random Patterns Generator
PUF	Physical Unclonable Function
RAM	Random Access Memory (assumed to be volatile)
RO	Read-Only
RW	Read/Write
SoC	System on Chip
SR	Security Requirement
TOE	Target Of Evaluation
UUID	Universal Unique IDentifier version 5 as defined in [RFC 4122]
WO	Write-Only

For the purposes of the present document, the following coding 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.
- Security Requirements established in this document are indicated by [SREQ**XX**] or by a dot in the SR column for tables in section 5.8.2. Both are used for easing their traceability.
- Functional Requirements established in this document are indicated by [REQ**XX**] in order to ease their traceability.
- Commands not explicitly indicated as “optional” are “mandatory”.

1.6 Revision History

Table 1-5: Revision History

Date	Version	Description
September 2017	0.1.0	Fast Track Member Review
November 2017	0.2.0	Fast Track Public Review
February 2018	0.3.0	Public Release Candidate
March 2018	1.0	Public Release

2 Overview

2.1 Objectives and Use Cases

The main objectives of the VPP are described in [IUICC Req]. They can be split as follows:

- Make the VPP Application as independent as possible from the underlying Primary Platform and consequently reduce the engineering effort to adapt the VPP Application to each different Primary Platform,
- Provide at least the same level of flexibility as the legacy secure platforms for supporting all use cases defined in [IUICC Req],
- Ease the VPP Application certification (if required) by enabling composite certification with a pre-certified Primary Platform.

The following Primary Platform specifications shall be provided by the Primary Platform Maker:

- The architecture of the CPU and its reference.
- The specification of the hardware functions accessible by the VPP Application[1].
- The LLOS ABI description adapting the interface to the LLOS implementation for a given CPU
- The Primary Platform dependent parameters defined in section 7

2.2 VPP Concept

The TRE contains three logical parts:

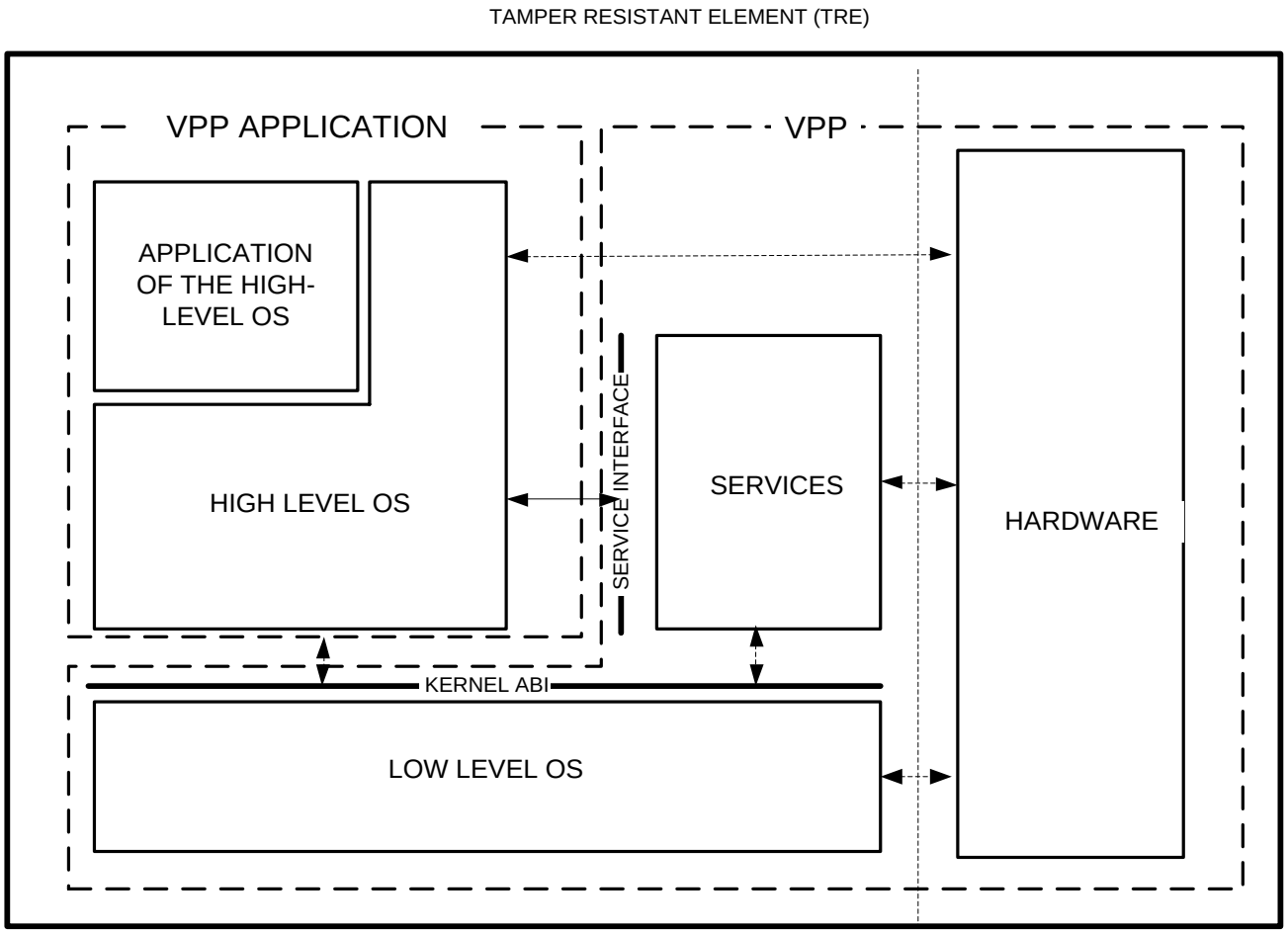
1. The Primary Platform consisting of the LLOS, its Processes and the hardware platform.
2. The abstraction and the Virtualization of the Primary Platform, called VPP, making[2] the interface to any Primary Platform virtually the same.
3. The use case dependent application of the VPP (i.e. a VPP Application) consisting of a HLOS and its application(s).

Figure 2-1 illustrates the internal architecture of the TRE.

[1] Via the VRE as defined in the section 5.7.4.5.

[2] It is foreseen that a Firmware needs, to a certain extent, to be adapted and re-compiled for the Primary Platform.

Figure 2-1: TRE Internal Architecture



From the VPP Application perspective, the VPP hides the Primary Platform.

2.3 The content of VPP Concepts and Interfaces

- The requirements of the Primary Platform
- The Virtual Primary Platform
- The virtualization of resources as described in the section 5
- The interface with the LLOS
- The interface with the Communication Service
- The interface with the Management Service
- The VPP Application
- The MLOI

3 Hardware Platform

3.1 Hardware Architecture

The TRE shall embed [SREQ**1**]:

- One or more CPU(s), which shall be specifically built for high-level of security
- Random Access Memory (RAM)
- Non Volatile Memory (NVM) of potentially different types (reprogrammable or not)
- A Random Number Generator Function
- Long-term credentials storage
- Security functions (e.g. sensors)
- A Key Protection function to restrict CPU access to secret key material
- A Memory Management Function in charge of translating the physical memory addresses to virtual memory addresses and protecting memory spaces according to their access rights.

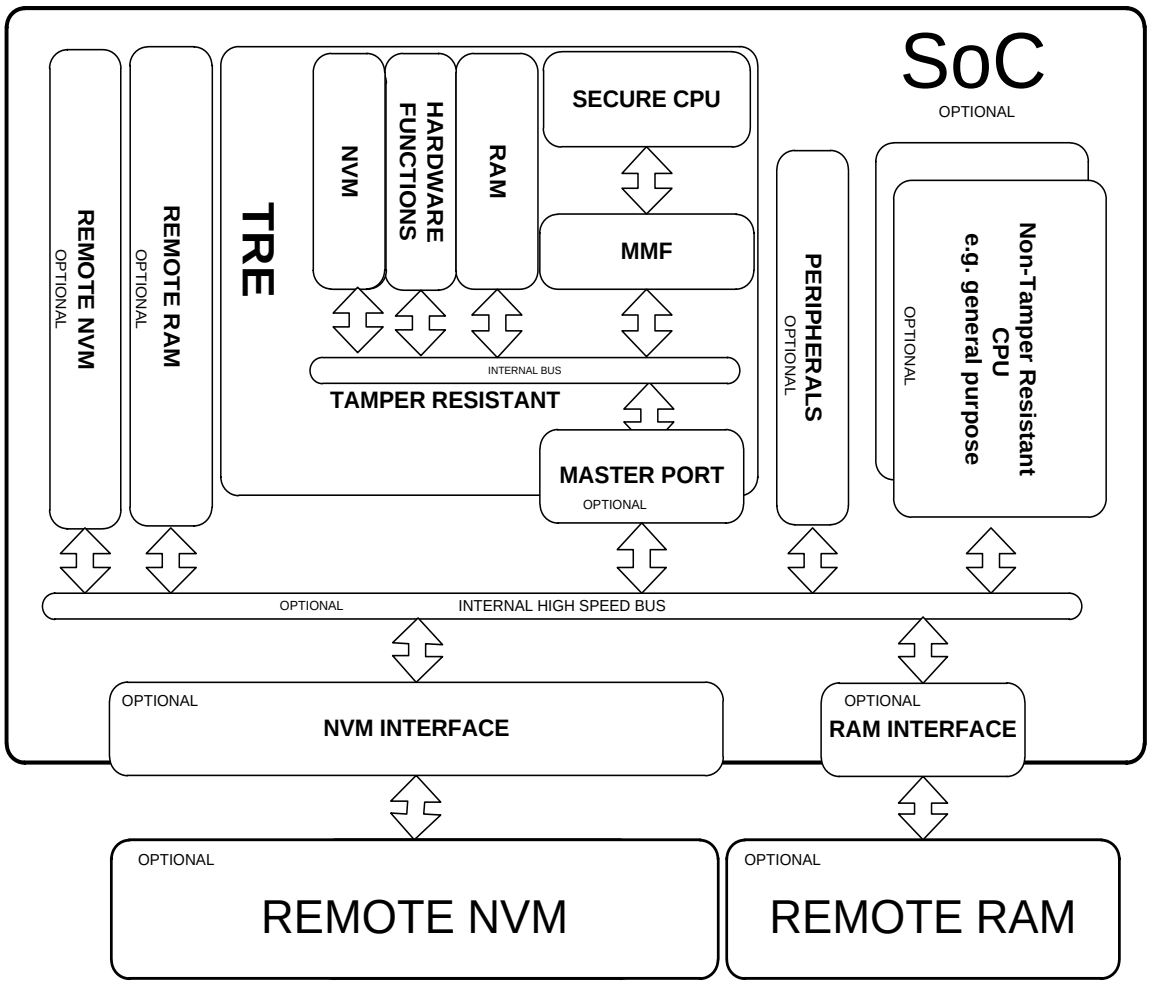
The hardware platform should furthermore embed [SREQ**2**]:

- Cryptographic Functions
- Memory Transfer Function (e.g. a Direct Memory Access controller)

Figure 3-1 shows a generic hardware platform for an Integrated TRE. For example, the SoC may be a mobile broadband modem, an application processor, a micro-controller or a dedicated controller (e.g. an NFC controller).

Figure 3-1shows a particular example, in the sense that the SoC may contain many additional elements that are not shown;

Figure 3-1: Example of Hardware Architecture



3.2 Generic Core features

3.2.1 CPU

The CPU shall support at least two execution modes: Privileged CPU Mode and Unprivileged CPU Mode [SREQ**3**].

The CPU architecture shall support an Application Binary Interface (ABI) able to transfer scalar parameters between a Program running in Unprivileged CPU Mode and a Program running in Privileged CPU Mode [SREQ**4**].

Note: 'CPU' in this document refers to the CPU inside the TRE that executes any Program in Primary Platform.

3.2.2 Memory Access

The Primary Platform shall provide a mean to filer memory access, for each CPU mode, based on a combination of multiple memory access (execute, read, write) [SREQ**5**].

3.2.3 Non Volatile Memory

At least one of the following options shall be implemented in the hardware platform:

- One Time Programmable NVM (OTP) [REQ**6**].
- Reprogrammable NVM [REQ**7**].
- Non-programmable NVM (i.e., ROM) [REQ**8**].

The hardware platform may use a remote NVM relying on a NVM outside the Security Perimeter of the TRE [REQ**9**].

3.2.4 Form Factor

No form factor is mandated.

3.2.5 Power

The overall power consumption of the Primary Platform is outside the scope of this specification. For this document, the Primary Platform shall be considered as powered up and capable of supporting the specified behavior. The Primary Platform Maker may implement power management strategies that may not support specified behavior. For example, power down or power collapse. Such strategies and their accompanying states are too outside the scope of this document.

3.2.6 Memory Transfer Function

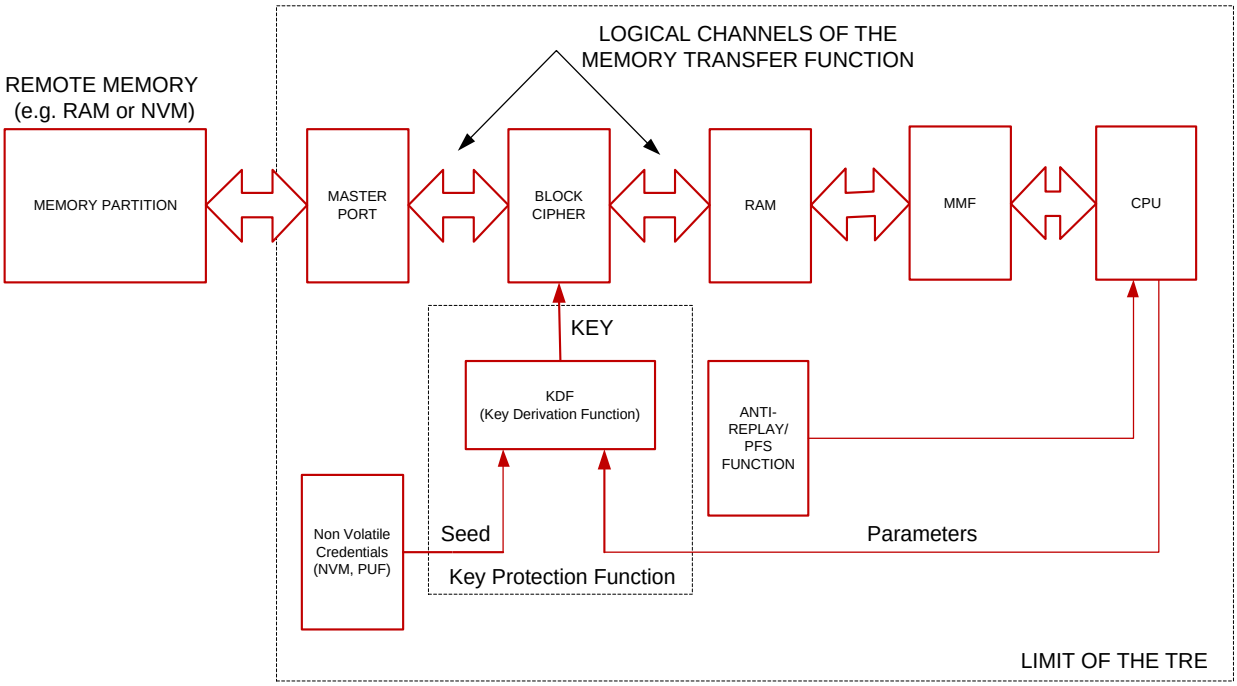
The hardware platform may provide a Memory Transfer Function (e.g. DMA) to allow a direct data transfer (meaning without the use of the CPU) between two resources (i.e. hardware interface to function or memory).

The Master Port is controlled by the TRE and makes the bridge between the TRE and the SoC buses.

For example, a transfer of data between the Remote Memory, via the interface of a function (e.g. Master Port), outside of the TRE, and a resource within TRE (e.g. block cipher).

Figure 3-2 illustrates the concept of memory transfer function.

Figure 3-2: Memory Transfer Function Example



In the above example, two logical channels of the memory transfer function allow the data transfer between a Remote Memory (i.e. either RAM or NVM) and a memory area located inside the TRE via a block cipher function.

If a memory transfer function, is required for transferring data outside the TRE then:

- The Remote Memory (e.g. Memory Partition) and the TRE memory area(s) shall be isolated by a hardware function (e.g., a master port) in order to ensure the following:
  - Access control is managed by the TRE [SREQ**10**],
  - Address translation across the different address spaces (e.g. the SoC address space and the TRE address space).

If the memory transfer function is required for transferring data inside or outside the TRE, the following requirement shall be fulfilled: The Primary Platform shall provide a means to protect against memory content access violations from hardware memory transfer [SREQ**11**].

Data stored in Remote Memory must be integrity protected and confidentiality protected [SREQ**12]**.

3.2.7 Cryptographic Functions

The TRE shall only execute cryptographic operations within its Security Perimeters (i.e., the TRE SP) [SREQ**13**]. The Primary Platform may support, the following cryptographic primitives through hardware assistance [REQ**14**]:

Table 3-1: Cryptographic Recommendations

Category	**Algorithm**
Block cipher	AES as defined in [FIPS 197]
Cryptographic hash	SHA-224, SHA-256, SHA-384, SHA-512 as defined in [FIPS 180-4]

	SHA-3 as defined in [FIPS 202]
Digital signature	ECDSA as defined in [FIPS 186-4]
Key exchange	ECC-based key exchange algorithms as defined in [RFC 4492]
Message authentication	HMAC with SHA-2: HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 as defined in [FIPS 198-1], [RFC 2104] and [RFC 4231] HMAC with SHA-3 as defined in [FIPS 202] AES-CMAC (CMAC with AES-128) as defined in [RFC 4493] GCM and GMAC as defined in [NIST SP 800-38D]
Message cipher	AES with confidentiality modes as defined in [NIST SP800-38A]

3.2.8 Random Number Generator Function

The TRE shall only execute random number generation operations within its Security Perimeters (i.e., TRE SP) [SREQ**15**]

The Random Number Generator Function should be in conformance with [SREQ**16**]:

- AIS20 certification for a DRNG (Deterministic Random Number Generator) as defined in AIS20 version 1 [AIS20].
- AIS31 certification for a TRNG (True Random Number Generator) as defined in AIS31 version 1 [AIS31].

3.3 System Functions

The hardware platform shall embed an autonomous and independent clock and reset system confined in the TRE Security Perimeter [SREQ**17**].

The hardware platform shall provide a tick counter, which has a fixed tick duration and is rising [REQ**18**].

The hardware platform may embed additional system functions (e.g., interrupt controller, etc.) out of the scope of this document.

3.4 Security Functions

3.4.1 General

The Primary Platform shall have an exclusive control over the mechanism that controls the access to the Primary Platform from outside the TRE [SREQ**19**].

The Primary Platform shall provide means for protecting itself against side channel attacks (hardware and software), Differential Power Analysis (DPA) and Differential Fault Analysis (DFA) attacks [SREQ**20**].

3.4.2 Memory Encryption and Integrity

The TRE shall only depend on its own internal cryptographic software and hardware functions [SREQ**21**].

The robustness of the memory encryption shall be equivalent to or greater than AES-256-CBC [SREQ**22**].

Memory encryption and integrity may be combined with a hardware integrity check [REQ**23**].

The robustness of the memory integrity shall be equivalent to or greater than HMAC-SHA-256 [SREQ**24**].

3.4.3 Key Protection Function

The hardware platform shall provide and use a Key Protection Function to protect specific long-term key material [SREQ**25**].

Keys used for encryption and integrity checking of Remote Memory shall be protected by the Key Protection Function and shall not be directly accessible by the CPU [SREQ**26**].

The Key Protection Function shall utilize a hardware key derivation function as defined in [NIST SP 800-57 Pt1 R4] section 8.2.4 [SREQ**27**]. The hardware key derivation function shall provide an interface that accepts [SREQ**28**]:

- A variable accessible only by Programs executing in Privileged CPU Mode,
- A diversified persistent secret seed (e.g. from a TRE NVM or from a PUF) which is only accessible by the Key Protection Function,

The Key Protection Function shall support a mode where its output shall be the key for the Cryptographic Functions [SREQ**29**].

3.4.4 Security Sensor Function

The hardware platform may embed security sensor function in charge to detect abnormal operating conditions. The implementation of such function is outside the scope of this document.

3.5 Memory Management Function

The Primary Platform shall embed a MMF under the control of the LLOS [SREQ**30**] for supporting [SREQ**31**]:

- The Forward Compliance property, regardless of the Firmware size.
- A Virtual Address Space, eliminating Processes dependency on Physical Address Space of the hardware platform.

The Memory Management Function shall enforce the access rules (i.e. Access Groups) of every Security Perimeters as defined in the section 5.2 [SREQ32].

3.6 Memory Storage Function

The memory storage area is considered as being remote when it is located outside the TRE. Multiple remote memory areas may be available with different persistency (e.g. ROM, RAM, OTP, NVM) and access times (e.g. reading, writing).

Regardless of the persistency of the memory (volatile or non-volatile in relation to the power cycles of the SoC), the term “cache” defines a memory storage area having a fast access time.

3.7 Remote Audit Function

For the sake of simplicity, this section considers the term VPP Application as either the System VPP Application or a VPP Application as defined in section 5.11.

Remote Audit Function works by allowing the VPP Application to provide the Remote Audit Function with value called ‘challenge’ and retrieving the result. The retrieved result can then be sent outside of the TRE, compared to a precomputed result, establishing integrity if both results are identical.

3.7.1 Remote Audit Function Requirements

The hardware platform should support a Remote Audit Function [SREQ**33**].

If the hardware platform supports the Remote Audit Function, then the requirements of this section shall apply.

In TOE life cycle phase 7 as defined in [PP-0084], the Remote Audit Function shall provide a means for a VPP Application to challenge the hardware platform against modifications by comparing it to the hardware platform that was submitted to certification laboratories [SREQ**34**].

The following requirements shall be fulfilled by the hardware platform:

- Support a Remote Audit Function which is capable of, with a probability greater than 0.8, to detect a hardware platform modification [SREQ**35**].
- Prohibit access to key material by the Remote Audit Function [SREQ**36**].
- Provide a means for a VPP Application to confidentially, with respect to the LLOS and any SoC sub-systems, a challenge into the Remote Audit Function [SREQ**37**].
- Provide a means for the VPP Application to confidentially retrieve the Remote Audit Function result [SREQ**38**].

Only a single[3] Remote Audit Function challenge may be injected into the Remote Audit Function by a given Firmware[4]; during the entire Firmware life cycle[5] [SREQ**39**].

The duration of Remote Audit Function operation[6] should not exceed MK_MAX_RAF_TIME_MS [REQ**40**].

The VPP Application session (as defined in section 6.3) shall be terminated to allow the Remote Audit Function operation to start [SREQ**41**]. The termination of the VPP Application session may be initiated by Main Process or by the MGT Process.

The following illustrates how the Remote Audit Function could be used to check that the TRE has not been modified after its certification:

- Once the compliance testing of a given TRE completes successful, the certification laboratory generates a certain number of challenges, to be fed into the Remote Audit Function. The results are then stored into a database[7].
- The remote auditor;
- Requests a set of archived challenge/result couples.
- Establishes a secure communication channel[8] with the VPP Application.

[3] The requirement prevents a Denial of Service attack by a VPP Application.

[4] Excluding the Firmware used for the instantiation of the system VPP Application.

[5] From its initial loading to its deletion.

[6] The protocol between the TRE and other SoC sub-systems that is used to properly stop the TRE, switch the TRE hardware platform into Remote Audit Function mode and then back to the operational mode is VPP implementation specific.

[7] The management of the database of the certification laboratory is out of the scope of this document.

[8] The procedure for setting up a secure communication between a remote auditor and a VPP Application may be proprietary and is out of scope of this document

- Transfer over the secure communication channel, an archived challenge to the VPP Application and collects the corresponding result from the Remote Audit Function over the secure communication channel.
- Compares the archived result corresponding to the collected result.

Any archived/collected result mismatch shall indicate that the hardware platform has been modified [SREQ**42**].

3.7.2 BIST Remote Audit Function option

3.7.2.1 Remote Audit Function based on BIST

The Remote Audit Function may use BIST[9] to challenge the hardware platform. The Remote Audit Function challenge is the seed of the PRPG. The hardware platform isolated from other SoC sub-system is the CUT. The Remote Audit Function result is the MISR output subsequent to a cryptographic hash function.

The archived challenge/result couples generated by the Remote Audit Function should not be predictable.

The number of possible Remote Audit Function challenge/result couples shall be large enough to be resistant to precomputation of all possible challenge/results couples.

3.8 Hardware Service Function

The Hardware Service Function are implementation dependent and out of the scope of this document.

One of the Hardware Service Functions is related to the communication between the TRE and the SoC sub-systems, e.g. physical layers.

[9] E.g., L-BIST (Logic BIST)

4 Primary Platform Certification

The certification of the Primary Platform shall claim in its Security Target one of the following options [SREQ**43**]:

- Conformance with Protection Profile BSI-CC-PP-0084-2014 [PP0084] [SREQ**44**]
- Conformance with Protection Profile BSI-CC-PP-0084-2014 [PP0084] and the certification by composition of the Loader package 2 [SREQ**45**].
- Conformance with Protection Profile BSI-CC-PP-0084-2014 [PP0084] including the Loader package 2 [SREQ**46**].

Note: Only the two last options are compliant with the requirements described in [iUICC Req].

The Primary Platform shall be encompassed by the certification Target of Evaluation (TOE) boundary [SREQ**47**].

The Security Target of the Primary Platform instance shall cover the security requirements listed in the present document [SREQ**48**].

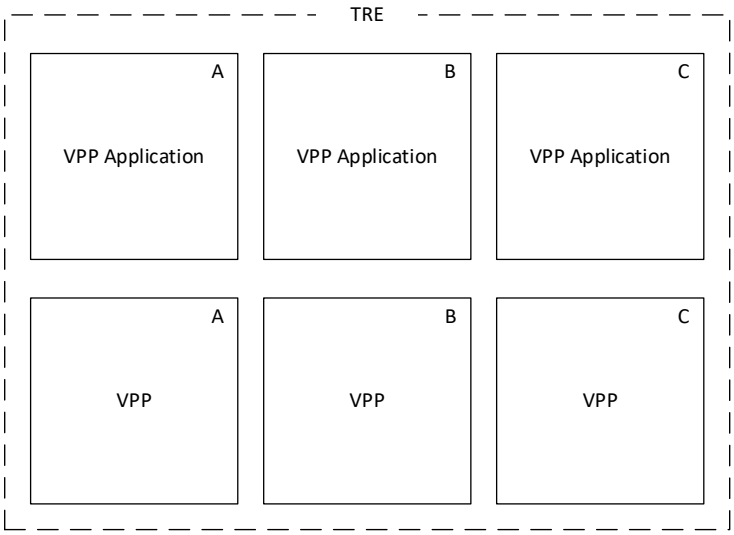
The certification minimum assurance level shall be at least EAL4 augmented with AVA_VAN.5 and ALC_DVS.2 [SREQ**49**].

AVA_VAN.5 tests should be performed in accordance with the JIL Application of Attack potential to Smartcards documentation [JIL] [SREQ**50**].

As illustrated in the Figure 4-1, a TRE may support multiple VPP instances as long as:

- Each VPP instance is compliant with the requirements set by the present document as well as both [VFF] and [VNP].
- A VPP Application running on a VPP instance shall not be confronted with any differences of behavior (e.g. timings), rules, security, certifications and MLOI when compared to running in a TRE supporting a single VPP instance.

Figure 4-1 : Multiple VPP instance in a TRE

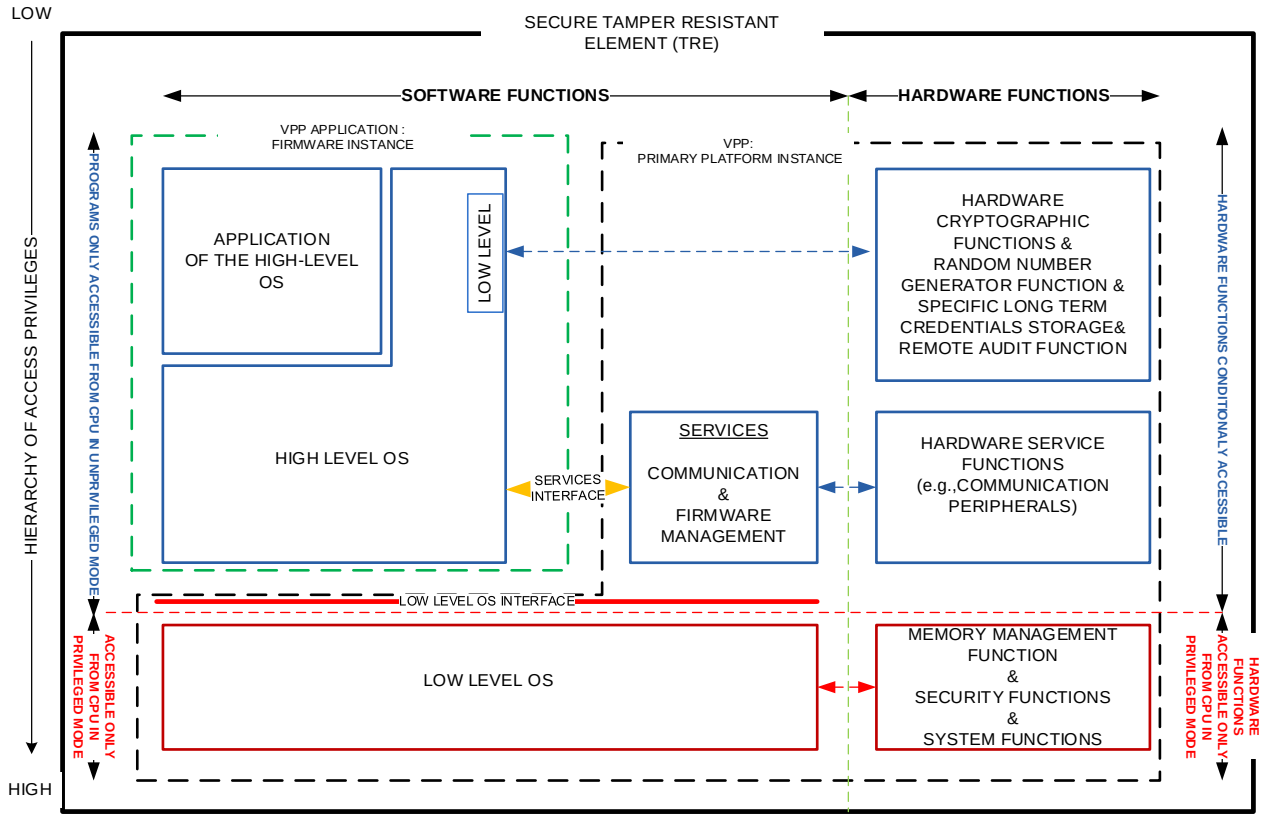


5 Virtual Primary Platform

5.1 Overview

Figure 5-1 describes the functional architecture for a TRE.

Figure 5-1: TRE Functional Architecture

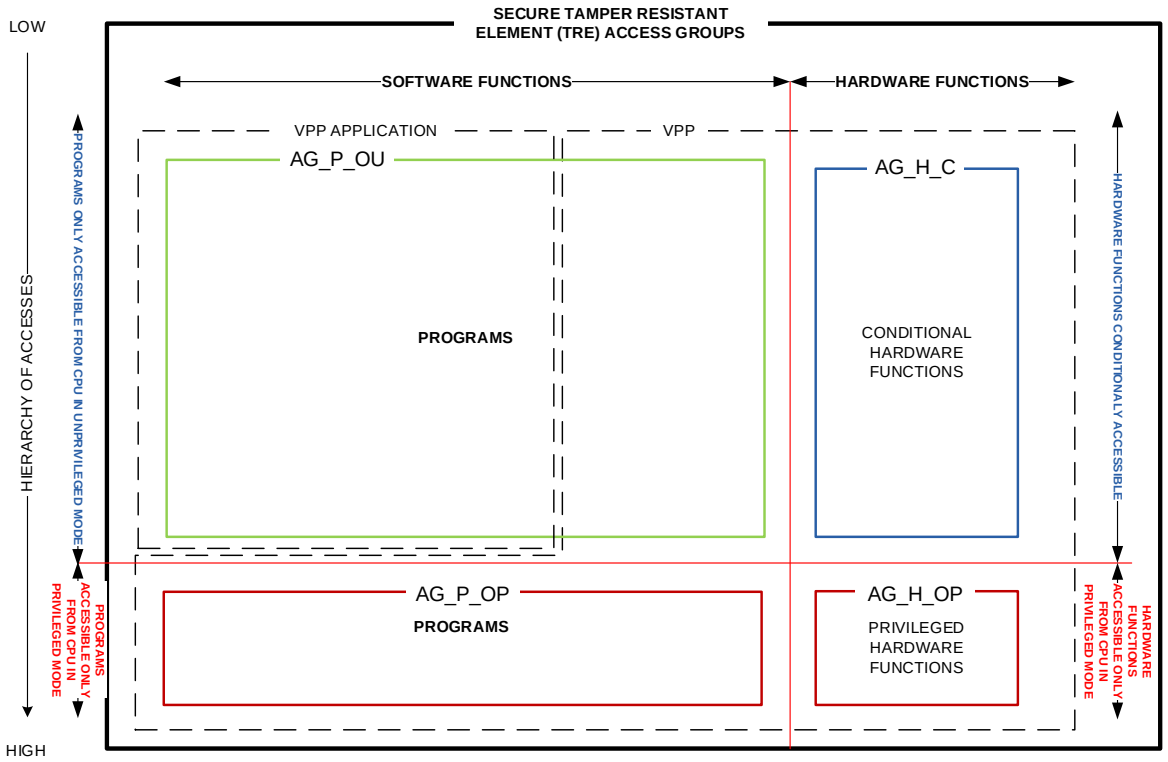


5.2 Access Groups

A combination of four Access Groups is defined within the TRE:

1. AG_P_OU: Any Program and data only accessible from a CPU running in unprivileged mode.
2. AG_H_C: Any hardware function conditionally accessible from the CPU (see sections 5.8.5.5 and 5.11).
3. AG_P_OP: Any Program and data only accessible from a CPU running in privileged mode.
4. AG_H_OP: Any hardware function only accessible from the CPU running in privileged mode

Figure 5-2: TRE Access Groups



The software functions are grouped in three domains:

- The LLOS managing security-related hardware functions and native multiprocessing capabilities.
- The HLOS (acting as a secondary platform) and its accompanying applications.
- The services managing the hardware functions related to communication (defined in section 5.9) and Firmware management (defined in section 5.10).

The hardware functions are grouped in two modules:

1. The Privileged Hardware Functions (AG_H_OP) which include:
   - System Functions
   - Security Functions
   - Memory Management Function
2. The Conditional Hardware Functions (AG_H_C) which include:
   - Hardware Service Functions
   - Cryptographic Functions

- Remote Audit Function
- Long-term credentials storage

The Primary Platform consists of the Hardware Platform, the Services within the VPP Execution Domain and the LLOS.

The abstraction and the virtualization of the Primary Platform is provided by the Virtual Primary Platform (VPP) which is made of three parts:

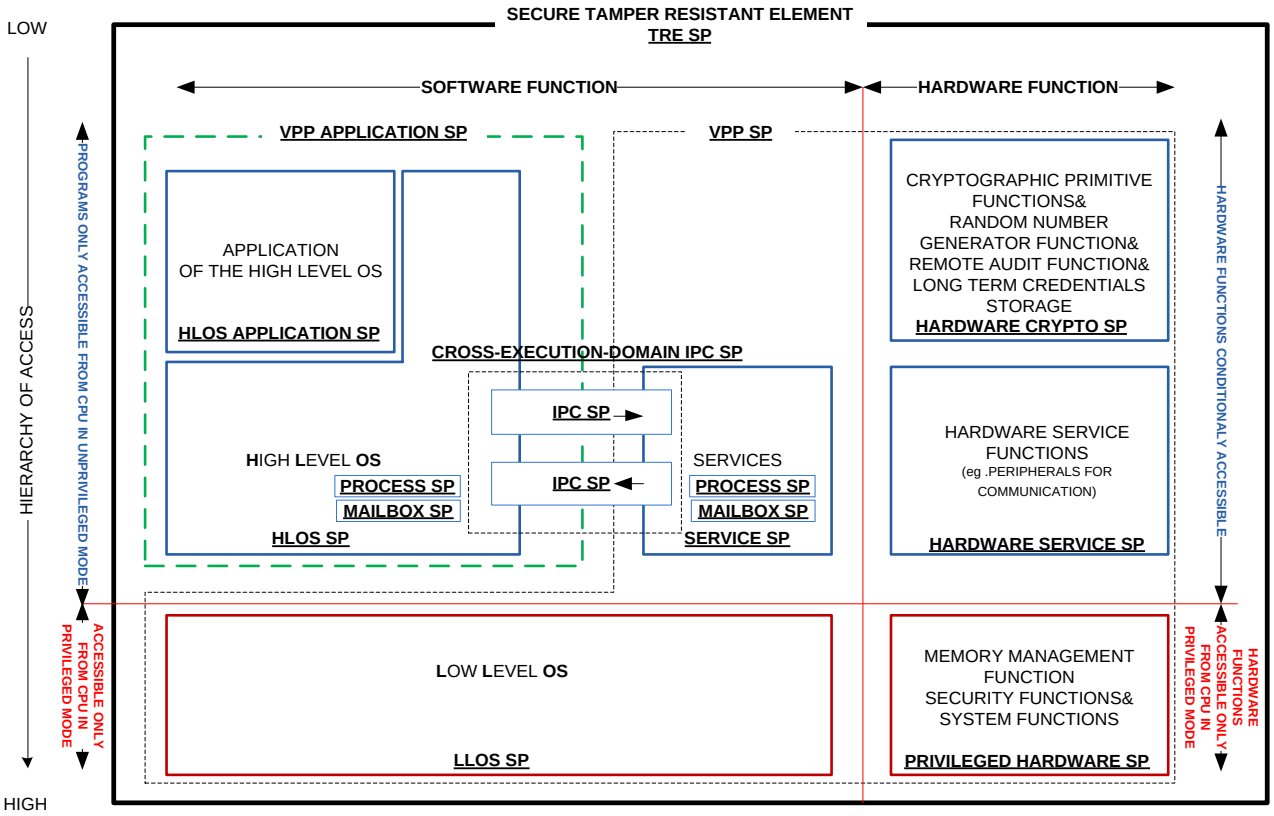
1. LLOS interface (described in section 5.8.5) and the MMF allowing the virtualization of the physical memory (described in section 5.5)
2. The service interfaces related to communication (described in section 5.9).
3. The service interfaces related to Firmware management (described in section 5.10).

5.3 Security Perimeters

The Security Perimeter (SP) defines the perimeter of a function on which rules, Access Groups, properties and requirements shall apply.

Figure 5-3 illustrates the SPs of the TRE.

Figure 5-3: TRE Security Perimeters



Functional description of the Security Perimeters:

- **VPP APPLICATION SP** in section 6,
- **HLOS APPLICATION SP** in section 6.4.1,

- **HLOS SP** in section 6.4,
- **VPP SP** in section 5,
- **CROSS-EXECUTIONDOMAIN IPC SP** in section 7.4 and 7.5,
- **SERVICE SP** in section 5.9 and 5.10,
- **HARDWARE SERVICE SP** in section 3.8,
- **HARDWARE CRYPTO SP** in section 3.2.7,
- **LLOS SP** in section 5.8,
- **PRIVILEGED HARDWARE SP** in section 3.3, 03.4, 3.5, 3.2.6 and 3.2.1,
- **PROCESS SP** in section 5.4, 5.5 and 0,
- **MAILBOX SP** in section 5.8.5.3,
- **IPC SP** in section 5.8.5.4,

The following requirements shall be fulfilled:

- **TRE SP** shall contain a single VPP APPLICATION SP at any given time [SREQ**51**]. The processing of TRE data shall be performed inside the TRE SP [SREQ**52**]. The storage of TRE data outside the TRE SP shall be protected for confidentiality, integrity, anti–rollback, software side channel attack and perfect forward secrecy by means solely located within the PRIVILEGED HARDWARE SP [SREQ**53**] The TRE code/data stored in the Remote Memory TRE SP shall be bound to the TRE [SREQ**54**].
- **VPP APPLICATION SP** shall: contain at least an HLOS APPLICATION SP, a HLOS SP and be in the Access Group AG_P_OU [SREQ**55**].
- **HLOS APPLICATION SP** shall be in the Access Group AG_P_OU [SREQ**56**].
- **HLOS SP** shall contain at least a PROCESS SP [SREQ**57**].
- **HLOS SP** may contain multiple IPC SP [SREQ**58**].
- **VPP SP** shall contain: a LLOS SP, a CROSS-EXECUTION-DOMAIN IPC SP, a PRIVILEGED HARDWARE SP, multiple IPC SP, a SERVICE SP, a HARDWARE CRYPTO SP and a HARDWARE SERVICE SP [SREQ**59**].
- **CROSS-EXECUTION-DOMAIN IPC SP** shall be able to transfer data to and from the SERVICE SP, from to HLOS SP. It shall be in the Access Group AG_P_OU [SREQ**60**].
- **SERVICE SP**: shall contain at least two PROCESS SPs called MGT and COM SP and be in the Access Group AG_P_OU [SREQ**61**].
- **SERVICE SP** may be able to transfer data to and from the HARDWARE SERVICE SP [SREQ**62**].
- **HARDWARE SERVICE SP** shall be in the Access Group AG_H_C [SREQ**63**].
- **HARDWARE CRYPTO SP** shall be in the Access Group AG_H_C [SREQ**64**].
- **LLOS SP** shall be: able to transfer data and credentials to and from the HARDWARE SYSTEM SP and shall be in the Access Group AG_P_OP [SREQ**65**].
- **PRIVILEGED HARDWARE SP** shall be in the Access Group AG_H_OP [SREQ**66**].
- **PROCESS SP** shall run a single Process and prevent data transfer to and from any SP except via a declared IPC SP. It shall be in the Access Group AG_P_OU [SREQ**67**].
- **PROCESS SP** may handle multiple MAILBOX SP [REQ**68**].

- **MAILBOX SP** shall only have a single PROCESS SP as receiver and have only a single PROCESS SP as sender and be either in the Access Group AG_P_OU or AG_P_OP [SREQ**69**].
- **IPC SP** shall: only contain a shareable memory space in the Access Group AG_P_OU, have only a single PROCESS SP as Accessor, have only a single PROCESS SP as Mutator, and be in the Access Group AG_P_OU [SREQ**70**].

The following rules shall apply to the TRE SP:

- Data transfer between Security Perimeters in the Access Group AG_P_OU and the Security Perimeter in the Access Group AG_P_OP shall occur using fixed, pre-determined CPU registers [SREQ**71**].
- Data transfer from Security Perimeters in the Access Group AG_P_OU to Security Perimeter in the Access Group AG_P_OP shall be restricted to Identifiers, Handles and Signals [SREQ**72**].
- Data transfer from Security Perimeters in the Access Group AG_P_OP to Security Perimeter in the Access Group AG_P_OU shall be restricted to memory address, Handle, Mailbox content, Errors and Exceptions [SREQ**73**].

5.4 Unprivileged Execution Model

A Process shall be always executed in unprivileged CPU mode [SREQ**74**]. Each Process shall have its own Virtual Address Space [SREQ**75**].

Each Process shall run on a Virtual Hardware Platform defined in section 6.1 [SREQ**76**].

A Process may implement a proprietary multithreading system managing its own Threads without the assistance of the LLOS[10].

5.5 Unprivileged Virtual Address Space

Any Process memory shall be mapped to the Virtual Address Space defined in Figure 5-4 [SREQ**77**]. The gap between the start locations of two adjacent Virtual Address Space Region is equal to MK_VSPACE_REGION_SIZE bytes.

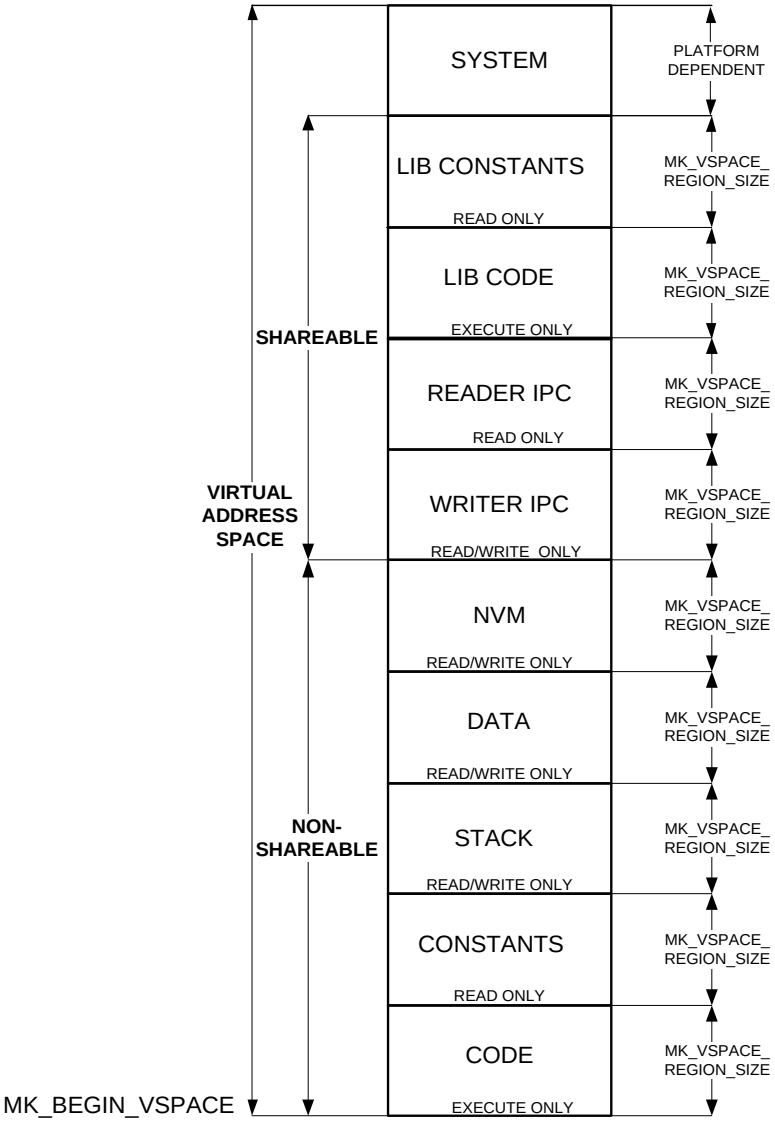
The size of the data within each Virtual Address Space Region, is defined in the Firmware, specifically in the Firmware Header, as defined in [VFF].

A memory access performed by a Process outside the boundaries of a sub Virtual Address Space is a Security Perimeter violation.

The stack of a Process may reside in the Physical Address Space and shall have the same access rights and boundaries protection as if it resided in Virtual Address Space [SREQ**78**].

[10] For example, the stack overflow protection is only available for the stack of the process. By implementing a proprietary multithreading system within a process, the designer of the multithreading system has no hardware assistance for stack overflow detection.

Figure 5-4: Virtual Address Space Mapping for unprivileged CPU mode



For each Process, the following regions are defined in Virtual Address Space:

- READER IPC: Used by IPC reader Process (read-only memory access).
- WRITER IPC: Used by IPC writer Process (read/write memory access)
- NVM: Persistent storage (read/write memory access)
- DATA: Initialized volatile storage (read/write memory access)
- STACK: Program stack (read/write permission)
- CONSTANTS: Process constants (read-only memory access)
- CODE: Process code (execute only memory access)
- SYSTEM: Reserved for the Primary Platform use; e.g. address for impersonation as defined in the section 5.8.5.6 (memory access according to use).
- LIB CONSTANTS: Constants for shared library (read only memory access)
- LIB CODE: Code for shared library (execute only memory access).

5.6 Run Time Model

The instance of the Primary Platform (i.e. VPP) and the instance of the Firmware (i.e. VPP Application) is made of two parts:

- The LLOS.
- A collection of Processes assigned to the Execution Domains.

Depending on their functions and their Execution Domain (VPP or VPP Application), a Process and the LLOS may have access to some specific hardware functions [REQ**79**].

Any Process shall only communicate with the LLOS by using the kernel ABI defined in section 5.8.5 [SREQ**80**].

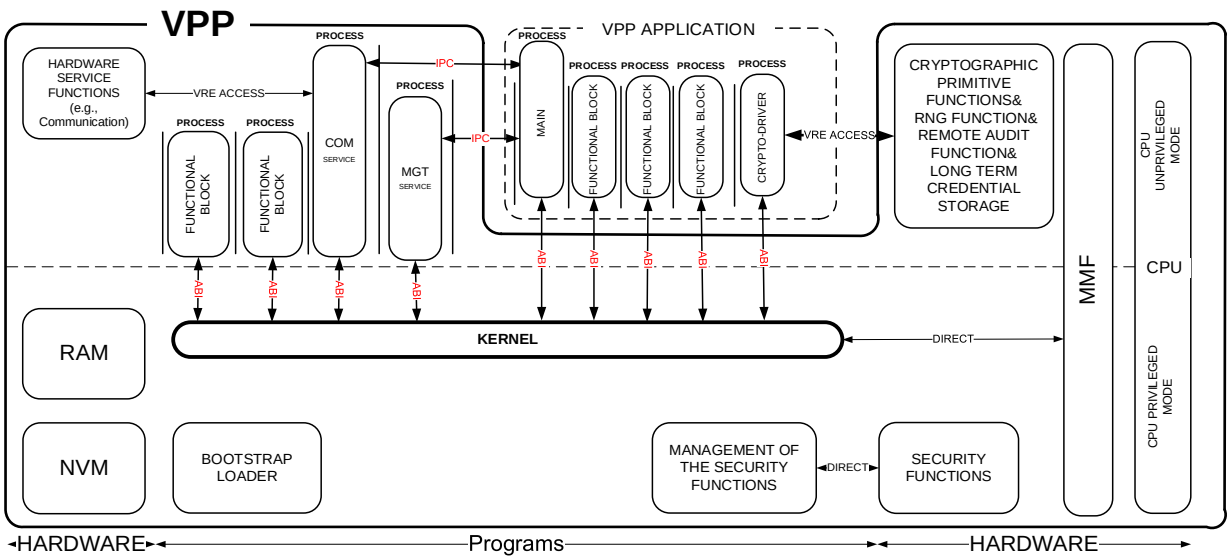
A Process shall only communicate with another Process by using IPC and Signals [SREQ**81**].

5.6.1 Exception Handling

Exceptions are reserved for severe run-time error that requires termination of the affected Process. If the Exception is not in the Main Process, then Main Process has a chance to shut down other Processes and then terminate itself. If the Exception is in the Main Process, the Main Process and all its descendants are terminated.

In order to recover a terminated VPP Application, it has to be restarted at the next VPP Application Session opening, as defined in section 6.3.

Figure 5-5: Runtime Model



5.7 Provisioning of Firmware and Primary Platform Software

The Primary Platform shall provide an interface for supporting the Firmware Loader [REQ**82**]. The Interface shall support a Firmware Loader as defined in [OFL] [SREQ**83**].

The Firmware Loader shall be the System VPP Application [SREQ**84**].

The Primary Platform may support provisioning of Primary Platform Software according to OFL [OFL] [REQ**85**].

5.8 Low Level Operating System (LLOS)

The Primary Platform embeds a LLOS running in privileged CPU mode. This LLOS is minimal and contains only the functionality which cannot run in unprivileged CPU mode [SREQ**86**].

The LLOS shall include a kernel supporting the management of multiple Processes [SREQ**87**].

In addition to a kernel, the LLOS shall support [SREQ**88**]:

- The initial Bootstrap Program of the Primary Platform,
- The management of all the following hardware functions:0
- Security Functions, as defined in section 3.4
- Memory Management Functions, as defined in section 3.5
- Memory Transfer Functions, as defined in section 3.2.6
- System Functions, as defined in section 3.3

5.8.1 Kernel Objects

The kernel manages multiple internal Kernel Objects:

- Mailbox for receiving Signals;
- IPC for communication between two Processes;
- Process for running a Program;
- VRE to enable a Process to directly access a hardware function.

In addition to the above Kernel Objects, the kernel manages the following notification mechanism:

- A Signal as a notification without additional data that may be sent from a Process to another Process or it may be sent from the kernel to a Process. A Signal is typically used to indicate that a pre-defined event has occurred. See 7.5 for a list of such pre-defined Signals.
- An Exception as a notification without additional data that is sent by the kernel to a parent Process when a special condition occurred in one of its child Processes. A severe Exception is thrown by the kernel when a Process has caused a violation that led to its termination by the kernel.

Kernel Objects are addressed by their owner Process using a Kernel Object Identifier. Some identifiers are shared between the Execution Domains are pre-defined by this document in the section 7.4. All other identifiers are defined by the Firmware Makers during Firmware design stage.

Kernel Object Handles are run-time identifiers for instantiated Kernel Objects. When operating on Kernel Objects, the Kernel ABI requires Kernel Object Handles. A Process shall retrieve Kernel Object Handles from the kernel by using Kernel Objects Identifiers [SREQ**89**]. A Process shall use Kernel Object Handles when interacting with the kernel in order to use Kernel Objects [SREQ**90**]. A Kernel Object Handle shall be valid only within the context of its owner Process [SREQ**91**]. The kernel shall restrict Kernel Object Handle use to the Process owning the object [SREQ**92**].

5.8.2 Global Requirements and Mandatory Access Control Rules

Table 5-1 defines the requirements for any Primary Platforms.

Table 5-1: Global Requirements

Rule	SR	Description
GR1	●	The Primary Platform shall provide a mechanism to guarantee the confidentiality of any data in Non-Shareable Memory spaces.
GR2	●	The Primary Platform shall provide a mechanism to guarantee the integrity of any data in Non-Shareable Memory spaces.
GR3	●	The Primary Platform shall provide a mechanism to ensure that access to a hardware function, including its input and output data, is exclusive and confidential to each Accessor or Mutator.
GR4	●	The Primary Platform shall provide a mechanism to restrict the access to hardware functions only to authorized Accessors or Mutators.
GR5	●	The LLOS shall only have a Non-Shareable Memory Space.
GR6	●	The requirements above shall be enforced by the LLOS.
GR7	●	The kernel shall be able to manage memory assigned to any Process.
GR8	●	The kernel shall communicate with a Process only using Signals and Kernel Service ABI using scalars as parameters (via registers).
GR9	●	The MGT Process is the parent of the Main Process.
GR10	●	VRE (Virtual Register) access shall be exclusive between the access and the release of the VRE.
GR11	●	A Process accessing the VRE should clean up the hardware function related to the VRE before releasing it.
GR12		For VPP Application Processes, collaborative scheduling shall be supported.
GR13		For VPP Application Processes, pre-emptive scheduling should be supported.
GR14		For VPP Application Processes, scheduling type shall be a declared option in its Firmware.
GR15		The Primary Platform shall only accept a Firmware if the Primary Platform capabilities meet the Firmware requirements, as described in its header.
GR16		For VPP Processes, scheduling shall be pre-emptive.
GR17	●	A VPP Process in ‘Waiting’ state may pre-empt a VPP Application Process in ‘Running’ state.
GR18	●	In case the collaborative scheduling is required, a pre-empted VPP Application Process shall be the next Process to execute.
GR19	●	VPP Processes shall have higher priority than VPP Application Processes.
GR20	●	A Process shall be instantiated in the “Suspended-R” state.
GR21	●	A Process shall be able to suspend itself or any Process in its sub-hierarchy.
GR22	●	A Process shall be able to resume any Process in its sub hierarchy.
GR23	●	When a Process dies, all its resources as well as resources owned by its sub-hierarchy Processes shall be released for future use by other Process. Confidentiality of the dead Process(es)’ resources must be maintained.

GR24	●	The MGT Process shall limit its control over the Main Process to its suspension and resumption.
GR25	●	All Kernel Objects (e.g. Mailboxes, IPC, VRE,) belonging to or used by a Process shall be instantiated before the Process is instantiated.
GR26	●	Accessing a non-existing Kernel Objects shall throw a severe Exception to the parent Process.
GR27	●	Any severe error (e.g. kernel rules infringement, memory firewall model violation) in a Process shall throw an Exception to the parent Process.
GR28	●	Any severe error in a Process in the “Running” state shall set the Process in the “Dead” state.
GR29	●	Exceptions shall be cleared by the kernel only after having been read by the parent Process.
GR30	●	The current VPP Application shall be terminated prior starting the next VPP Application.
GR31	●	All resources (Processes, Mailboxes, IPC, VRE) related to the VPP Application shall be allocated during the Firmware instantiation.
GR32	●	The Handle of a Kernel Object provided to a Process shall only be valid/used for that Process.
GR33	●	The kernel shall throw a severe Exception[11] to the parent Process of a Process which violates its Security Perimeter.
GR34	●	The kernel shall reject unknown or undefined kernel calls by throwing a severe Exception.
GR35	●	Any severe Exception in MGT Process shall reset the TRE.
GR36		VPP shall provide VPP Applications a monotonic and rising tick counter during VPP Application Session.

Table 5-2 defines the access rules to resources, granted to Processes.

Table 5-2: Mandatory Access Control Rules

Rule	SR	Description
AC1		Access to a Service, a LLOS interface or to a Hardware Function shall be denied unless explicitly allowed.
AC2		MAC rules shall be conjunctive.
AC3		The MAC rules described in this document shall be the most permissive. Primary Platform Makers and Firmware Makers may reduce the required access to kernel calls and resources; permitting only resources and kernel calls needed by a VPP Application or available in the Primary Platform.
AC4	●	A (Writer) Process shall define one or more IPCs for which that Process shall have

[11] MK_EXCEPTION_SEVERE as defined in the section 7.3.

		read/write-access. These IPCs shall be owned exclusively by that Process.
AC5	●	A (Reader) Process shall define one or more IPCs for which it shall have read-only access. These IPCs shall not be owned by that Process.
AC6	●	An IPC shall only accept a single Writer Process and a single Reader Process.
AC7	●	A (Receiver) Process shall declare one or more Mailboxes from which that Process shall be able to receive Signals. These Mailboxes shall be owned exclusively by that Process.
AC8	●	A (Sender) Process shall declare one or more Mailboxes to which that Process may send Signals. These Mailboxes shall not be owned by that Process.
AC9	●	A Mailbox shall only accept Signals from a single source, either a Process or the kernel.
AC10	●	Every Process shall have a specific Mailbox to which only the kernel may send Signals. This Mailbox shall be owned by the Process, and the Process shall be able to receive Signals.
AC11	●	Only the owner of a Mailbox shall be able to read its content

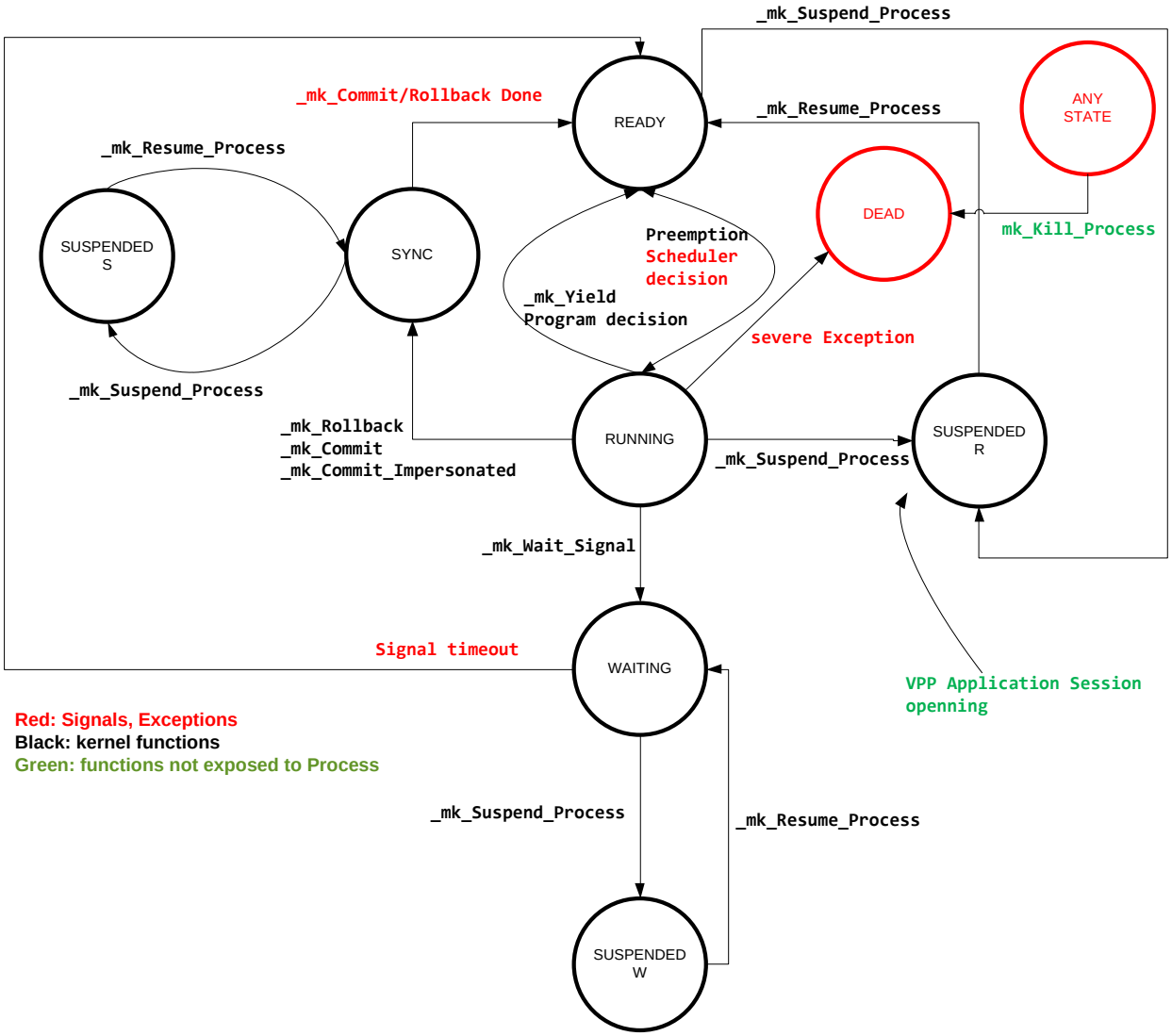
Note: All Mailboxes and IPCs shall be defined during VPP Application development and in its resulting Firmware.

5.8.3 Process States Diagram

Figure 5-6 illustrates the state diagram of a Process. An event (IN or OUT) allows a Process to change states. Events may be:

- Call of a Kernel ABI function (e.g. _mk_Yield),
- A Signal from a Process via the calling of _mk_Send_Signal function,
- A Signal from the kernel related to a VRE,
- An Exception,
- A timeout,
- A preemption by the kernel scheduling function.

Figure 5-6: Process State Diagram



This state diagram is applicable to both VPP and VPP Application Processes. Only one Process may be in the “Running” state at any given time.

5.8.4 Definition of the Process States

Processes in 'Suspended' states (R/S/W), may receive Signals, but will handle the Signals once the Process is resumed [REQ**93**].

When a Process transition between different states, the scheduler shall place the Process as the last position is the queue assigned to the new state with the exception of GR17 [REQ**94**].

The scheduler shall select Processes for operations based first on Process priority and then in order in the Process queue [REQ**95**].

Table 5-3 defines in detail the states of a Process.

Table 5-3: Definition of States

State Name	Description	Event in	Event out
Ready	The Process is eligible for running.	_mk_Resume_Process, Signal, time-out, _mk_Yield Completion of mk_Commit or _mk_Rollback Scheduler decision, e.g. a VPP Process has received a Signal.	Scheduler decision, _mk_Suspend_Process
Running	The Process is running.	Selected for execution by the scheduler	_mk_Yield, _mk_Suspend_Process, _mk_Wait_Signal, Scheduler decision: e.g. a Signal was sent to another Process[12], _mk_Commit, _mk_Rollback, _mk_Commit_Impersonated Severe Exception
Sync	The Process is blocked until _mk_Commit or _mk_Rollback is completed.	_mk_Commit, _mk_Rollback _mk_Commit_Impersonated _mk_Resume_Process	Completion of _mk_Commit or _mk_Rollback, _mk_Suspend_Process
Waiting	The Process is waiting for a Signal or an elapsed timeout.	_mk_Wait_Signal _mk_Resume_Process	Signal (via _mk_Wait_Signal), time-out, _mk_Suspend_Process

[12] The process preemption occurs in one of the followings cases:

- The Process of the VPP Execution Domain sends a Signal to another Process of the same Execution Domain which has a higher priority.
- The VPP Application has declared in its Firmware the use of preemptive scheduling and the Process in its Execution Domain sends a Signal to another Process of the same Execution Domain which has a higher priority.
- The VPP Application has declared in its Firmware the use of a collaborative scheduling and the process in its Execution Domain sends a Signal to a process of the VPP Execution Domain.

Suspended R	The Process remains in the “Suspended R” state until it is resumed by its parent Process.	_mk_Suspend_Process Process instantiation (Swap IN VPP Application)[14]	_mk_Resume_Process
Suspended S	The Process remains in the “Suspended S” state until it is resumed by its parent Process. When resumed, the Process is in the “Sync” state.	_mk_Suspend_Process	_mk_Resume_Process
Suspended W	The Process is in “Suspended W” state until it is resumed by its parent Process. When resumed, the Process is appended to the end of the list of Processes in the” Waiting” state.	_mk_Suspend_Process	_mk_Resume_Process
Dead	The Process is no longer active due to a termination, a severe Exception, a rule violation or a violation of the memory firewall model.	Process termination[13] Severe Exception	Firmware instantiation[14]

[13] Internal events are expressed for easing the reading of the table but are not exposed in the ABI

[14] Firmware instantiation: All Kernel Objects of a Firmware are instantiated (i.e. Process, Mailboxes, IPC, VRE) and the Memory Partition containing the Firmware are mounted and selected. Prior to a Firmware instantiation on the VPP the previous Firmware instance is removed.

5.8.5 Kernel Functions ABI/API

The Primary Platform Maker shall provide an ABI (Application Binary Interface) related to the implementation of the LLOS in order to map any API (Application Programming Interface) which are dependent of the programming language made available on the HLOS [REQ**96**].

The Primary Platform Maker shall provide the C language API using the above ABI and mapping it to the C function prototypes as defined in this section [REQ**97**].

All data types and constant values are defined in chapter 7.

5.8.5.1 Generic Functions

5.8.5.1.1 Function _mk_Get_Exception

Brief: Retrieve an Exception.

Description: This function retrieves the last Exception thrown by the kernel. Exceptions are cleared after reading. Only exceptions of child processes can be retrieved.

Parameter:

- _hProcess (MK_HANDLE_t) Process Handle

Return:

- bitmap (MK_BITMAP_t) A MK_EXCEPTION_e bitmap where each bit represents a unique Exception, as defined in Table 7-11.

C prototype function:

MK_BITMAP_t _mk_Get_Exception(MK_HANDLE_t _hProcess)

5.8.5.1.2 Function _mk_Get_Error

Brief: Get the last error generated through the execution of a function within a given Process. The returned error value is volatile.

Description: This function retrieves an error stored by kernel. The access to the last error is always possible for a Process and any of its descendants regardless of its state and persistent during state transitions (see Table 5-3).

Parameter:

- _hProcess (MK_HANDLE_t) Handle of the Process

Return:

- error (MK_ERROR_e) value of the error (see Table 7-12).

C prototype function:

MK_ERROR_e _mk_Get_Error(MK_HANDLE_t _hProcess)

5.8.5.1.3 Function _mk_Get_Time

Brief: Get the absolute time (in ticks) since the Primary Platform start up.

Description: The return value is 64 bits in length. It is important to note that whenever the Primary Platform starts or restarts, the timer is reset to zero. The returned value represents elapsed time only during the caller's VPP Application Session. Between different VPP Application Sessions there is no guarantee on elapsed time or even for the value being monotonic and increasing.

Parameter:

- None No parameters

Return:

- time (MK_TIME_t) value of the current time in ticks

C prototype function:

MK_TIME_t _mk_Get_Time(void)

5.8.5.2 Process Management

5.8.5.2.1 Function _mk_Get_Process_Handle

Brief: Get the Process kernel Handle for itself or for one of its descendants.

Description: This function gets a Process kernel Handle through its Process identifier.

The Process retrieving the Process Handle does not inherit the rights of its owner.

Parameter:

- _eProcess_ID (MK_PROCESS_ID_u) identifier of the Process

Return:

- Handle (MK_HANDLE_t) Handle of the Process Kernel Object.
- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_UNKNOWN_ID if the ID does not exist or the Process is in "Dead" state.
  - MK_ERROR_ACCESS_DENIED if the Process is not one of the descendants of the caller Process.

C prototype function:

MK_HANDLE_t _mk_Get_Process_Handle(MK_PROCESS_ID_u _eProcess_ID)

5.8.5.2.2 Function _mk_Get_Process_Priority

Brief: Get the Process priority.

Description: This function gets the priority of a Process.

Parameter:

- _hProcess (MK_HANDLE_t) Handle of the Process

Return:

- Priority (MK_PROCESS_PRIORITY_e) Priority of the Process or the priority reserved for error
- MK_PROCESS_PRIORITY_ERROR On error _mk_Get_Error function returns the following error code:
  - MK_ERROR_UNKNOWN_HANDLE if the Process Handle is invalid.

C prototype function:

MK_PROCESS_PRIORITY_e _mk_Get_Process_Priority(MK_HANDLE_t _hProcess)

5.8.5.2.3 Function _mk_Set_Process_Priority

Brief: Set the Process priority.

Description: This function sets the priority of a Process. A Process may change its priority or the priority of one of its descendants. The priority can be changed anytime, independent from the state.

Parameter:

- _hProcess (MK_HANDLE_t) Handle of the Process
- _xPriority (MK_PROCESS_PRIORITY_e) Priority of the Process

Return:

- MK_ERROR_NONE if the function is successful
- MK_ERROR_UNKNOWN_HANDLE if the Process Handle is invalid
- MK_ERROR_UNKNOWN_PRIORITY if the priority value is invalid

C prototype function:

MK_ERROR_e _mk_Set_Process_Priority(MK_HANDLE_t _hProcess, MK_PROCESS_PRIORITY_e _xPriority)

5.8.5.2.4 Function _mk_Suspend_Process

Brief: Suspend a Process. A Process can suspend itself or any of its descendants.

Description: This function suspends a Process. The suspended Process is no longer scheduled for execution. If a Process suspends itself, then this call will only return upon resumption by the parent Process.

Parameter:

- _hProcess (MK_HANDLE_t) Handle of the Process to be suspended

Return:

- MK_ERROR_NONE if the Process suspended successfully
- MK_ERROR_UNKNOWN_HANDLE if the Process Handle is invalid
- MK_ERROR_ACCESS_DENIED if the Process is not itself or any of its descendants.

C prototype function:

MK_ERROR_e _mk_Suspend_Process(MK_HANDLE_t _hProcess)

5.8.5.2.5 Function _mk_Resume_Process

Brief: Resume a Process

Description: This function resumes a Process. A resumed Process must be a descendant of the running Process.

Parameter:

- _hProcess (MK_HANDLE_t) Handle of the Process

Return:

- MK_ERROR_NONE if the Process resumes successfully
- MK_ERROR_UNKNOWN_HANDLE if the Process Handle is invalid
- MK_ERROR_ACCESS_DENIED if the Process is not one of its descendant

C prototype function:

MK_ERROR_e _mk_Resume_Process(MK_HANDLE_t _hProcess)

5.8.5.2.6 Function _mk_Request_No_Preemption

Brief: Allows a VPP Application Process to request a period of time during which it cannot be pre-empted by a VPP Process.

Description: This call requests continuous CPU processing allocation to a given Process for a certain amount of time and should help handling operations that require more exact timing. Requests with a period of time shorter than MK_APP_STOP_GRACEFUL_TICKS shall be ignored except when the requested time value is 0, which will resume pre-emption This function is optional.

Parameter:

- _uTime _____(uint32_t) requested time shall not exceed MK_NO_PREEMPTION_MAX_TIMEOUT (number of ticks). If uTime is 0 then pre-emption becomes possible.

Return:

- MK_ERROR_NONE if the request is accepted
- MK_ILLEGAL_PARAMETER if _uTime exceeds MK_NO_PREEMPTION_MAX_TIMEOUT

C prototype function:

MK_ERROR_e _mk_Request_No_Preemption(uint32_t _uTime)

5.8.5.2.7 Function _mk_Commit

Brief: Commits all the changes in the caller's NVM.

Description: This function commits all changes to the NVM of the caller Process, as defined in [VFF]. No roll-back is possible after calling this function. The caller Process is suspended until the completion of this operation. This operation is atomic and cannot fail (unless due to an irrecoverable error).

Parameter:

- void No parameters

Return:

- void No returned value.

C prototype function:

void _mk_Commit(void)

5.8.5.2.8 Function _mk_RollBack

Brief: Rolls back all the changes made to the caller's NVM.

Description: This function rolls back all changes to the NVM of the caller Process, back to the last commit operation. The caller Process is suspended until the completion of this operation. This operation is atomic and cannot fail (unless due to an irrecoverable error).

Parameter:

- void No parameters.

Return:

- void No returned value.

C prototype function:

void _mk_RollBack(void)

5.8.5.2.9 Function _mk_Yield

Brief: Return the control to the kernel scheduler.

Description: Let the caller Process ask the kernel to yield its execution, causing the kernel to switch the caller to “Ready” state. This call will return when the Process is scheduled to run by the scheduler.

Parameter:

- void None.

Return:

- void No value returned.

C prototype function:

void _mk_Yield(void)

5.8.5.3 Mailbox Management

5.8.5.3.1 Function _mk_Get_Mailbox_Handle

Brief: Get a Mailbox Handle from a Mailbox identifier.

Description: This function gets the Mailbox Handle through a Mailbox identifier.

Parameter:

- _eMailboxID (MK_MAILBOX_ID_u) identifier of the Mailbox

Return:

- Handle (MK_HANDLE_t) of the Mailbox.
- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_ACCESS_DENIED if the Process is not allowed to send a Signal to the Mailbox
  - MK_ERROR_UNKNOWN_ID if the Mailbox identifier is invalid.

C prototype function:

MK_HANDLE_t _mk_Get_Mailbox_Handle(MK_MAILBOX_ID_u _eMailboxID)

5.8.5.3.2 Function _mk_Get_Mailbox_ID_Activated

Brief: When waiting for Signal on any Mailbox owned by the caller Process, get the Mailbox identifier of a Process that has a pending Signal.

Description: This function retrieves the identifier of a Mailbox with a pending signal when the Process waits on any Mailbox of the caller Process.

Parameter:

- void No parameter

Return:

- identifier (MK_MAILBOX_ID_u) of the Mailbox identifier
- NULL if no Mailbox has received a Signal

C prototype function:

MK_MAILBOX_ID_u _mk_Get_Mailbox_ID_Activated(void)

5.8.5.3.3 Function _mk_Send_Signal

Brief: Send a Signal to a Mailbox.

Description: This function sends Signals to a Mailbox. The signals sent are represented as a bitmap of Signal values and there is no priority among Signals as to the order of their arrival within the Mailbox.

Parameter:

- _hMailbox (MK_HANDLE_t) Handle of the Mailbox
- _eSignal (MK_BITMAP_t) Signal value

Return:

- MK_ERROR_NONE the Signal(s) was/were sent successfully
- MK_ERROR_UNKNOWN_HANDLE if the Mailbox Handle is invalid
- MK_ERROR_ACCESS_DENIED if the caller Process is not defined as the sender Process of the Mailbox

C prototype function:

MK_ERROR_e _mk_Send_Signal(MK_HANDLE_t _hMailbox, MK_BITMAP_t _eSignal)

5.8.5.3.4 Function _mk_Wait_Signal

Brief: Wait for a Signal on a Mailbox.

Description: This function waits for a Signal on one or any Mailboxes of the caller Process, either for given time or without a time limit. This call is blocking and will return when a signal is received or when the timeout occurred.

When a Process waits on any Mailbox, the Signals MK_SIGNAL_TIME_OUT, MK_SIGNAL_ERROR and MK_SIGNAL_EXCEPTION are sent only to its kernel Mailbox.

When a Process waits on a Mailbox, the Signals MK_SIGNAL_TIME_OUT, MK_SIGNAL_ERROR and MK_SIGNAL_EXCEPTION are sent to that Mailbox.

Only the owner of the Mailbox can wait on it.

Parameter:

- _hMailbox (MK_HANDLE_t) Handle of the Mailbox. If the Handle is NULL, then the Process shall wait for any Signal sent to any of the Mailboxes owned by the Process
- _uTime (uint32_t) timeout time in ticks.
  - If the value is 0, the function will not wait for a Signal and will return control to the caller Process immediately.

- If the value is MK_ENDLESS then the function will wait for a Signal forever and control will not be returned until a Signal is received.

Return:

- void no value is returned.

C prototype function:

void _mk_Wait_Signal(MK_HANDLE_t _hMailbox, uint32_t _uTime)

5.8.5.3.5 Function _mk_Get_Signal

Brief: Get a Signal from a Mailbox.

Description: This function gets a Signal on a Mailbox. A Process can only retrieve the Signal from its own Mailbox. The _mk_Get_Signal should be repeatedly called until 0 is returned. The pending Signals are cleared once they have been read.

Parameter:

- _hMailbox (MK_HANDLE_t) Mailbox Handle

Return:

- bitmap (MK_BITMAP_t) a bitmap where each bit represents a Signal (MK_SIGNAL_e)
- NULL On error _mk_Get_Error function returns the following error code:
  - 1<<MK_SIGNAL_ERROR if the Mailbox Handle is invalid, or the access is denied.

C prototype function:

MK_BITMAP_t _mk_Get_Signal(MK_HANDLE_t _hMailbox)

5.8.5.4 IPC Management

The content of the IPC owned by a VPP Application is persistent until the state of the VPP Application is instantiated.

5.8.5.4.1 Function _mk_Get_IPC_Handle

Brief: Get the Handle of an IPC.

Description: This function gets an IPC Handle for communication between two Processes.

The size, the ownership and the granted access of the IPC are defined in the IPC descriptor, which is part of the Firmware header.

The owner Process (i.e. writer) of the IPC has a read-write access.

The granted access Process (i.e. reader) has read-only access.

Parameter:

- _eIPC_ID (MK_IPC_ID_u) identifier of the IPC

Return:

- Handle (MK_HANDLE_t) IPC Handle
- NULL On error _mk_Get_Error function returns the following error code:
  - MK_ERROR_UNKNOWN_ID if the IPC identifier is not defined or if access is not allowed.

C prototype function:

MK_HANDLE_t _mk_Get_IPC_Handle(MK_IPC_ID_u _eIPC_ID);

5.8.5.4.2 Function _mk_Get_Access_IPC

Brief: Get access to a shared memory area used by an IPC.

Description: This function returns the virtual memory address of the IPC (virtual shared memory).

The number of concurrent access of the IPC is guaranteed to be at least limited MK_MIN_CONCURRENT_IPC_LIMIT.

A Process can only access the IPC virtual memory address if it is the owner or the granted Process, as described in the IPC descriptor

Parameter:

- _hIPC (MK_HANDLE_t) IPC Handle

Return:

- Virtual memory address of the IPC.
- NULL On error _mk_Get_Error function returns one of the following error code:
  - MK_ERROR_UNKNOWN_HANDLE if the IPC Handle is invalid
  - MK_ERROR_ACCESS_DENIED if the Process is not the allowed to access the IPC.

C prototype function:

void* _mk_Get_Access_IPC(MK_HANDLE_t _hIPC)

5.8.5.4.3 Function _mk_Release_Access_IPC

Brief: Release access to the IPC.

Description: This function allows releasing the access to the IPC. The Process cannot longer access the virtual shared memory.

The IPC is a scarce resource, thus the number of access of IPC is limited (MK_IPC_LIMIT) at the run time.

Parameter:

- _hIPC (MK_HANDLE_t) IPC Handle.

Return:

- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_NONE if the IPC releases successfully
  - MK_ERROR_UNKNOWN_HANDLE if the IPC Handle is invalid
  - MK_ERROR_ACCESS_DENIED if the Process is allowed to access the IPC

C prototype function:

MK_ERROR_e _mk_Release_Access_IPC(MK_HANDLE_t _hIPC)

5.8.5.5 VRE Management

5.8.5.5.1 Function _mk_Get_VRE_Handle

Brief: Get the Handle of a virtual register.

Description: This function gets the VRE Handle for communication with a hardware function. A VRE is used for direct access to a hardware function.

The access policy to VREs is defined in the Process Descriptor.

Parameter:

- _eVRE_ID (MK_VRE_ID_e) identifier of the VRE

Return:

- Handle (MK_HANDLE_t) VRE Handle.
- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_UNKNOWN_ID: if the VRE ID is invalid/unknown
  - MK_ERROR_ACCESS_DENIED: if the VRE violates the MAC as defined section 5.11 or exclusive with respects to another VPP Applications (e.g. Remote Audit Function)

C prototype function:

MK_HANDLE_t _mk_Get_VRE_Handle(MK_VRE_ID_e_eVRE_ID)

5.8.5.5.2 Function _mk_Get_Access_VRE

Brief: Allows the caller Process to get the address of a VRE.

Description: This function gets the virtual registers base address of the hardware function (VRE) to be accessed. The operation of said hardware function is Primary Platform specific, and dependent on the hardware used and exposed by the Primary Platform Maker.

The number of different VREs that can be accessed simultaneously by a VPP Application is limited to MK_VRE_LIMIT. If that limit is exceeded, then access to new VREs shall be denied.

Parameter:

- hVRE (MK_HANDLE_t) VRE Handle

Return:

- Virtual registers base address of the hardware function to access.
- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_UNKNOWN_HANDLE if the VRE Handle is invalid
  - MK_ERROR_ACCESS_DENIED if the number of VREs in use by the VPP Application has exceeded the MK_VRE_LIMIT or if the VRE is accessed from another Process

C prototype function:

void* _mk_Get_Access_VRE(MK_HANDLE_t _hVRE)

5.8.5.5.3 Function _mk_Release_Access_VRE

Brief: Release access to a VRE.

Description: This function releases the access to a VRE.

The Process is responsible to clean up the content of the hardware function before releasing the VRE, according to the specifications of the specific hardware function.

Parameter:

- _hVRE (MK_HANDLE_t) VRE Handle

Return:

- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_NONE if the VRE is released successfully
  - MK_ERROR_UNKNOWN_HANDLE if the VRE Handle is invalid

C prototype function:

MK_ERROR_e _mk_Release_Access_VRE(MK_HANDLE_t _hVRE)

5.8.5.5.4 Function _mk_Attach_VRE

Brief: Attach a VRE to the kernel Mailbox of a Process.

Description: Set the callback Signals to send to the caller Process kernel Mailbox when the status of a hardware function related to a VRE changes.

If a Process waits on Signals from an attached VRE and the attached VRE is preempted by another Process, then the MK_EXCEPTION_VRE_DETACHED Exception is thrown to the waiting Process.

This function is valid only if the Process declares the VRE access within the Process descriptor and if the use of the VRE is allowed by the Mandatory Access Control.

Parameter:

- _hVRE (MK_HANDLE_t) VRE Handle
- _uSignal (MK_BITMAP_t) value of the Signal(s) to send to the kernel Mailbox when the status of the VRE changes. The Signal(s) shall be in the range of MK_SIGNAL_DOMAIN_BASE_0 to MK_SIGNAL_DOMAIN_BASE_28 (inclusive).

Return:

- MK_ERROR_NONE if the VRE attachment is accepted
- MK_ERROR_UNKNOWN_HANDLE if the Handle is invalid or the Process Descriptor does not define a VRE access
- MK_ILLEGAL_PARAMETER if the provided bitmap is not in the allowed range of Signals.

C prototype function:

MK_ERROR_e _mk_Attach_VRE(MK_HANDLE_t _hVRE, MK_BITMAP_t _uSignal)

5.8.5.6 Firmware Management

The following functions apply to the Management Service Interface in section 5.10.

5.8.5.6.1 Function _mk_Open_Impersonation

Brief: Inform the kernel that the impersonation of a Firmware is started.

Description: This function can only be used by the System VPP Application (e.g. OFL).

This function works in conjunction with _mk_Impersonate_Process, which must be called in order to enable the writing to individual sub Memory Partition defined in [VFF]. When impersonation is done, this function needs to be followed with _mk_Close_Impersonation.

_mk_Open_Impersonation allows reading and writing in the Memory Partition of the registered Firmware.

Parameter:

- _uFirmwareID (UUID_t) Identifier of the Firmware to impersonate in [VFF]

Return:

- MK_ERROR_NONE if the impersonation is successfully opened
- MK_ERROR_UNKNOWN_UUID if the given _uFirmwareID is unknown
- MK_ERROR_INTERNAL if an internal error occurred, e.g., an impersonation was already started
- MK_ERROR_ACCESS_DENIED if the VPP application does not have the access rights

C prototype function:

MK_ERROR_e _mk_Open_Impersonation(UUID_t _uFirmwareID)

5.8.5.6.2 Function _mk_Close_Impersonation

Brief: Inform the kernel that the impersonation of a Firmware has completed.

Description: This function can only be used by the System VPP Application.

Parameter:

- void No parameters

Return:

- MK_ERROR_NONE if the impersonation is successfully closed

C prototype function:

MK_ERROR_e _mk_Close_Impersonation(void)

5.8.5.6.3 Function _mk_Impersonate_Process

Brief: Inform the kernel that the caller wishes to impersonate another Process' sub Memory Partition, e.g. CODE, belonging to the Firmware being loaded or updated.

Description: This function allows the System VPP Application to impersonate a sub Memory Partition of the Firmware being loaded or updated, so that the code, constants, data and the NVM areas of a sub Memory Partition may be written.

This function can only be used by the System VPP Application assigned at the Firmware loading.

Parameter:

- void No parameters

Return:

- The virtual memory address of the beginning of the impersonated sub Memory Partition, as defined in section 5.5 and marked as MK_BEGIN_VSPACE.

- NULL On error _mk_Get_Error function returns one of the following error codes:
  - MK_ERROR_UNKNOWN_ID if the PROCESS ID is invalid/unknown within the impersonated application
  - MK_ILLEGAL_PARAMETER if the segment index is invalid within the impersonated Process
  - MK_ERROR_ACCESS_DENIED if no Firmware impersonation has been performed prior calling _mk_Impersonate_Process

C prototype function:

void* _mk_Impersonate_Process(void)

5.8.5.6.4 Function _mk_Commit_Impersonated

Brief: Commit all changes in impersonated sub Memory Partition to NVM.

Description: This function allows the System VPP Application to commit the sub Memory Partition related to an impersonated Process. The System VPP Application may impersonate a shared library or the LLOS, if the Primary Platform supports this capability. No roll-back is possible after calling this function. The caller Process is suspended until the completion of this operation. This operation is atomic and cannot fail (unless due to an irrecoverable error).

Parameter:

- void No parameters

Return:

- void No returned value.

C prototype function:

void _mk_Commit_Impersonnated (void)

5.9 Communication Service Interface

The Communication service manages two data chunk FIFO queues as defined in 6.1 between the VPP (COM Process) and a VPP Application (Main Process):

- FIFO OUT as a FIFO queue allows transferring data chunks in sequence from a source Process (Main or COM) to a destination Process (COM or Main) containing arrays of m_Size_OUT data chunks each being m_MTU_OUT bytes long.
- FIFO IN as a FIFO queue allows transferring data chunks in sequence from a source Process (COM or MAIN) to the destination Process (Main or COM) containing Array of m_Size_IN data chunks of m_MTU_IN bytes.

The data chunk shall contain a packet as defined in [VNP] [REQ**98**].

The transfer of data is based on two IPC identified as:

- MK_IPC_COM_MAIN_ID for the data transfer from the COM Process to the Main Process.
- MK_IPC_ MAIN_COM_ID for the data transfer from the Main Process to the COM Process.

Both Processes using the IPC, obtain access to the IPC by retrieving the virtual memory addresses using the kernel function _mk_Get_Access_IPC.

For clarity, the virtual memory addresses used by the IPC between VPP Application Main Process and COM Process are:

- pIPC_COM_MAIN – The COM (Writer) ➔ Main (Reader) IPC.
- pIPC_MAIN_COM – The Main (Writer) ➔ COM (Reader) IPC.

The virtual shared memory address for transferring data from one Process to another are:

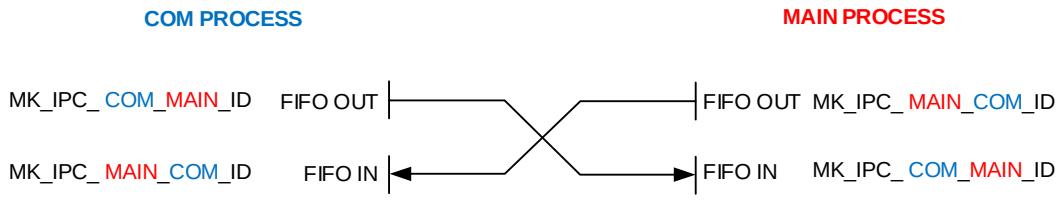
- FIFO OUT COM to Main: pIPC_COM_MAIN from the COM to the Main Process
- FIFO OUT Main to COM: pIPC_MAIN_COM from the Main to the COM Process

The FIFO OUT of the source Process is the FIFO IN of the destination Process:

- FIFO IN Main from COM: pIPC_COM_MAIN from the COM to the Main Process,
- FIFO IN COM from Main: pIPC_MAIN_COM from the Main to the COM Process.

Figure 5-7 illustrates the links between the FIFO IN and FIFO OUT.

Figure 5-7: FIFO IN and OUT links



The destination shall read its FIFO IN when the Signal MK_SIGNAL_IPC_UPDATED is sent by the source Process [REQ**99**].

5.9.1 FIFO Update procedure

Brief: Update the FIFO IN and OUT irrespective of the Process (COM or Main). The source Process shall inform the destination Process that its FIFO OUT has been updated.

The returned values are updated each time the Process receives the MK_SIGNAL_IPC_UPDATED Signal.

Description:

The parameters from the source Process Main or COM shall be filled with a structure pointed by pIPC_MAIN_COM or pIPC_COM_MAIN respectively.

The source Process shall send, for each update made to its FIFO OUT, an MK_SIGNAL_IPC_UPDATED Signal to the destination Process Mailbox.

The destination Process shall only read its FIFO IN after an MK_SIGNAL_IPC_UPDATED Signal has been received on the destination Process Mailbox [REQ**100**].

The command performs the following operations:

- The source Process signals the destination Process, using the MK_SIGNAL_IPC_UPDATED signal, that it wrote a data chunk.
- The location of the written data chunk is in m_Buff_OUT array at the index (m_Write_OUT-1) modulo m_Size_OUT.

- The destination Process signals the source Process, using the MK_SIGNAL_IPC_UPDATED signal, that it read a data chunk.
- The location of the read data chunk is in m_Buff_IN array at the index (m_Write_IN-1) modulo m_Size_IN.

Parameters:

- m_MTU_OUT (uint16_t) The FIFO OUT Maximal Transport Unit (size of a data chunk)
- m_Size_OUT (uint16_t) The FIFO OUT maximal number of data chunks
- m_Read_IN (uint32_t) The index of the last data chunk read in the FIFO IN
- m_Write_OUT (uint32_t) The index of the next data chunk to be written into the FIFO OUT
- m_Buff _OUT (array) Array of m_Size_OUT data chunks of m_MTU_OUT bytes

Return:

- m_MTU_IN (uint16_t) The FIFO IN Maximal Transport Unit (size of a data chunk)
- m_Size_IN (uint16_t) The FIFO IN maximal number of data chunks
- m_Read _OUT (uint32_t) The index of the last data chunk read in the FIFO OUT
- m_Write_IN (uint32_t) The index of the next data chunk to be written into the FIFO IN
- m_Buff_IN (array) Array of m_Size_IN data chunks of m_MTU_IN bytes
- The fields in the FIFO structure are packed in IPC memory, so there is no alignment and no padding.
- The algorithm supporting the management of both FIFO queues is the following:
- FIFO OUT is empty if m_Read_{OUT} equals $m\_Write_{OUT.}$
- FIFO IN is empty if m_Read_{IN} equals $m\_Write_{IN.}$
- FIFO OUT is full if ($m\_Write_{OUT.}$ - m_Read_{OUT}) >= m_Size_{OUT}
- FIFO IN is full if ($m\_Write_{IN.}$ - m_Read_{IN}) >= $m\_Size_{IN.}$
- The next data chunk to be written in FIFO OUT is m_Buff[m_Write_{OUT}]. The field m_Write_{OUT} shall be incremented after the writing of the data chunk.
- The next data chunk to be read in FIFO IN is m_Buff[m_Read_{IN}]. The field m_Read_{IN} shall be incremented after the reading of the data chunk. The content of the data chunk shall be considered as consumed, by the writer, as soon as m_Read_{IN} is incremented.

The indexes of the FIFO are unsigned 32-bit integers therefore the incrementing of the indexes is modulo 2^{32}. That leads to a wrong detection of a FIFO queue full when the algorithm encounters an arithmetic overflow. The following algorithm shall be applied [REQ**101**].

$Count_{XX}$ is an unsigned integer.

Compute $Count_{XX}$ ($m\_Write_{XX.}$ - m_Read_{XX}) is the number of pending data chunks in the FIFO xx.

IF $Count_{XX}$ < 0 THEN

$Count_{XX}$ = complement of $Count_{XX}$+1

END IF

5.10 Firmware Management Service Interface

This Service, running in the MGT Process, may support the management of multiple Firmwares regardless of their type (i.e. normal or system).

Firmware Management Service shall only be accessible by the system VPP Application [SREQ**102**].

The System VPP Application shall be responsible to either fully write or pad with zeros ('00') impersonated Memory Partition, according to its specified size defined in the Firmware header [SREQ**103**].

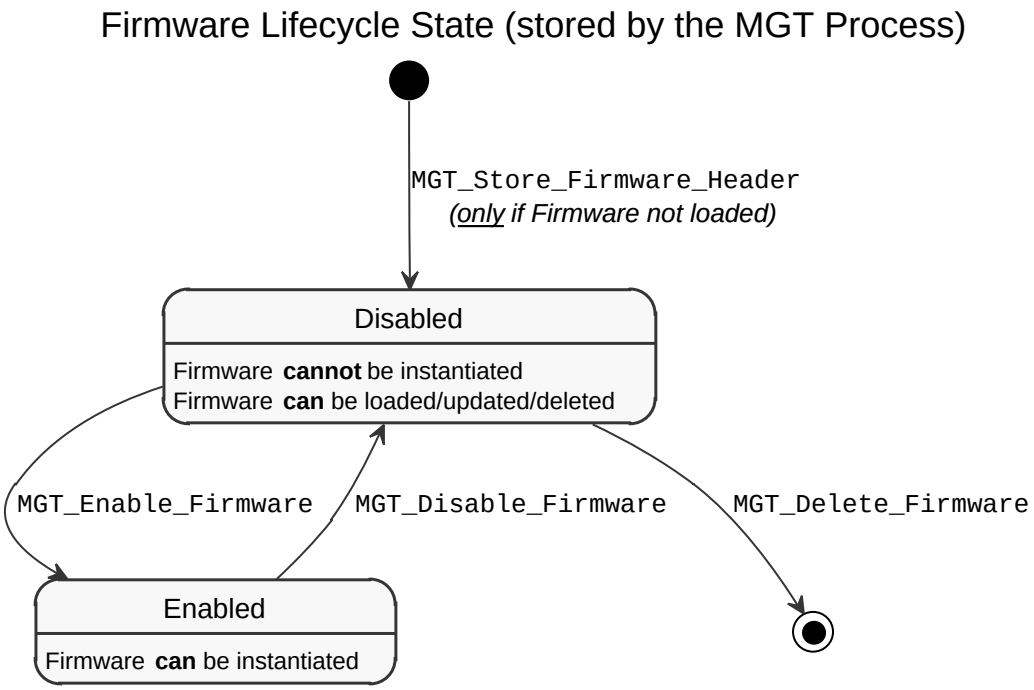
A VPP Application is an instance of a Firmware; a Firmware has two states:

- A Firmware in Enabled state may be instantiated [REQ**104**].
- A Firmware in Disabled state shall not be instantiated [SREQ**105**].

Firmware state shall be persistent against power cycles and is managed by the MGT Process [SREQ**106**].

Figure 5-8 illustrates the lifecycle state diagram in conjunction with the Management Service functionality.

Figure 5-8: Firmware Lifecycle State Diagram



When a command is sent from the Main Process to the MGT Process, the Main Process shall:

1. Fill a data structure related to the command to be executed and map it on the IPC MK_IPC_MAIN_MGT_ID [REQ**107**].
2. Send the Signal MK_SIGNAL_IPC_UPDATED to the Mailbox MK_MAILBOX_MAIN_MGT_ID [REQ**108**].
3. Wait for the Signal MK_SIGNAL_IPC_UPDATED on the Mailbox MK_MAILBOX_MGT_MAIN_ID [REQ**109**].
4. Read the response from a data structure through the IPC MK_IPC_MGT_MAIN_ID [REQ**110**].

When a response is to be returned from the MGT Process to the Main Process, the MGT Process shall:

1. Wait for the Signal MK_SIGNAL_IPC_UPDATED on the Mailbox MK_MAILBOX_MAIN_MGT_ID [REQ**111**].

2. Read the command data through the IPC MK_IPC_MAIN_MGT_ID [REQ**112**].
3. Send the response data through the IPC MK_IPC_MGT_MAIN_ID [REQ**113**].
4. Send the Signal MK_SIGNAL_IPC_UPDATED to the Mailbox MK_MAILBOX_MGT_MAIN_ID [REQ**114**].

Table 5-4 lists the commands.

Table 5-4: Management Service Commands

Command Code	Command
‘00’	MGT_Store_Firmware_Header
‘01’	MGT_Retrieve_Firmware_Header
‘02’	MGT_Allocate_Firmware
‘03’	MGT_Delete_Firmware
‘04’	MGT_Enable_Firmware
‘05’	MGT_Disable_Firmware
‘06’	MGT_Is_Firmware_Enabled
‘07’	MGT_Open_Process_Impersonation
‘08’	MGT_Close_Process_Impersonation
‘09’	MGT_Open_Library_Impersonation (optional)
‘0A’	MGT_Close_Library_Impersonation (optional)
‘0B’	MGT_Open_LLOS_Impersonation (optional)
‘0C’	MGT_Close_LLOS_Impersonation (optional)

Table 5-5 lists the response codes.

Table 5-5: Management Service Response Codes

Response Code	Definition
‘00’	MGT_ERROR_NONE
‘01’	MGT_ERROR_ILLEGAL_PARAMETER
‘02’	MGT_ERROR_INTERNAL
‘03’	MGT_ERROR_UNKNOWN_UUID
‘04’	MGT_ERROR_COMMAND_NOK

Table 5-6: Management Service Command /Response Codes Assignment

RESPONSE / COMMAND	MGT_Store_Firmware_Header	MGT_Retrieve_Firmware_Header	MGT_Allocate_Firmware	MGT_Delete_Firmware	MGT_Enable_Firmware	MGT_Disable_Firmware	MGT_Is_Firmware_Enabled	MGT_Open_Process_Impersonation	MGT_Close_Process_Impersonation	MGT_Open_Library_Impersonation	MGT_Close_Library_Impersonation	MGT_Open_LLOS_Impersonation	MGT_Close_LLOS_Impersonation
MGT_ERROR_NONE	●	●	●	●	●	●	●	●	●	●	●	●	●
MGT_ERROR _ILLEGAL_PARAMETER	●	●	●	●	●	●	●	●		●			
MGT_ERROR_INTERNAL			●	●	●	●	●	●	●	●	●	●	●
MGT_ERROR _UNKNOWN_UUID		●	●	●	●	●	●	●		●			
MGT_ERROR _COMMAND_NOK	●	●	●	●	●	●	●	●	●	●	●	●	●

Note: MGT_ERROR_COMMAND_NOK is reserved for the Primary Platform Maker, as a generic error.

The fields in the command and response structures are neither aligned nor padded.

5.10.1 Firmware Header Management

5.10.1.1 MGT_Store_Firmware_Header

Brief: Store the Firmware header as defined in [VFF].

Description:

The command performs the following operations:

- Parse the firmware_header
- If parsing failed
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Compare the firmware_header to the Primary Platform capabilities
- If firmware_header is not supported
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Extract the Firmware Identifier (UUID) from firmware_header

- Retrieve the firmware_header from the MGT Process NVM, based on its provided Firmware Identifier as defined in [VFF]
- If the firmware_header cannot be found
  - Add a new Firmware header, store the provided Firmware Identifier as the key to this record. The initial Firmware state should be 'Disabled'
- Else
  - Update firmware_header record
- Return response_code = MGT_ERROR_NONE

Parameters:

- 00 (uint8_t) MGT_Store_Firmware_Header Command.
- (firmware_header data) header as described in [VPP]

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.1.2 MGT_Retrieve_Firmware_Header

Brief: Retrieve from the MGT Process as defined in [VFF]

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code =MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM, that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- Return
  - response_code = MGT_ERROR_NONE
  - (the Firmware Header in the response)

Parameters:

- '01' (uint8_t) MGT_Retrieve_Firmware_Header command.
- firmware_identifier (UUID_t) identifier of the Firmware (ie. m_xName in [VFF]).

Return:

- response_code (uint8_t) management service response code as described in Table 5-5
- (Firmware Header data) The data of the Firmware Header, as defined in [VFF]

5.10.2 Firmware State Management

5.10.2.1 MGT_Enable_Firmware

Brief: Change Firmware state to ‘Enabled’.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM, that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- Instruct the COM Process that the identified Firmware shall be registered with the entity managing the communication to the TRE, may now receive, and be instantiated upon incoming data as defined in [VNP].
- Update the Firmware state in NVM belonging to MGT Process to ‘Enabled’
- If error while enabling Firmware
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- ‘04’ (uint8_t) MGT_Enable_Firmware command
- firmware_identifier (UUID_t) identifier of the Firmware (ie. m_xName in [VFF])

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.2.2 MGT_Disable_Firmware

Brief: Change Firmware state to ‘Disabled’.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM,that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID

- Instruct the COM Process that the identified Firmware shall be deregistered with the entity managing the communication to the TRE, shall not receive incoming data as defined in [VNP], and shall not be instantiated.
- Update the Firmware state in NVM belonging to MGT Process to 'Disabled'
- If error while disabling Firmware
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '05' (uint8_t) MGT_Disable_Firmware command.
- firmware_identifier (UUID_t) identifier of the Firmware (ie. m_xName in [VFF]) to be disabled

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.2.3 MGT_Is_Firmware_Enabled

Brief: Query if the state of a Firmware is 'Enabled'.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM, that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- Return
  - response_code = MGT_ERROR_NONE
  - state =
    - TRUE if the state of the Firmware is 'Enabled'
    - FALSE otherwise

Parameters:

- '06' (uint8_t) MGT_Is_Firmware_Enabled command.
- firmware_identifier (UUID_t) identifier of the Firmware (i.e. m_xName in [VFF]) to query

Return:

- response_code (uint8_t) management service response code as described in Table 5-5
- state (uint8_t) '01' if the Firmware is enabled, '00' if the Firmware is disabled

5.10.2.4 MGT_Delete_Firmware

Brief: Deletion of an existing Firmware related to a VPP Application.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- If Firmware state is not 'Disabled'
  - Return response_code = MGT_ERROR_INTERNAL
- Erase the Memory Partition from non-volatile data storage (NVM) and from any cache memory, based on the provided firmware_identifier
- If error while erasing
  - Return response_code = MGT_ERROR_INTERNAL
- Erase the Firmware Header and the Firmware state from the MGT Process NVM, based on the provided firmware_identifier
- Unregister the Firmware Identifier within the COM Process
- Return response_code = MGT_ERROR_NONE

Parameters:

- '03' (uint8_t) Code corresponding to MGT_Delete_Firmware command
- firmware_identifier (UUID_t]) identifier of the Firmware (i.e., m_xName in [VFF]) to be deleted

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3 Firmware Impersonation Management

5.10.3.1 MGT_Open_Process_Impersonation

Brief: Prepare the Primary Platform to impersonate a Process belonging to the Firmware being impersonated.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM,that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID

- Load the Memory Partitions of the Firmware
- Initiate the instantiation of a Kernel Object for the impersonation
- Initialize the above Kernel Object with the physical memory address of the sub-Memory Partition, belonging to the impersonated Process, which is itself part of the Firmware being loaded
- If error while impersonating
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '07' (uint8_t) MGT_Open_Process_Impersonation command.
- firmware_identifier (UUID_t) identifier of the Firmware (i.e., m_xName in [VFF])
- index (MK_Index_t) index of the Process Descriptor index within the array of Process Descriptors of the Firmware header defined in [VFF]

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3.2 MGT_Close_Process_Impersonation

Brief: Prepare a Firmware for closing the impersonation of a Process.

Description:

The command performs the following operations:

- Instruct the kernel to clear the reference related to the sub-partition for a Process to impersonate
- If error while closing impersonation
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '08' (uint8_t) MGT_Close_Process_Impersonation command.

Return:

- response_code (uint8_t) response in Table 5-5

5.10.3.3 MGT_Open_Library_Impersonation

Brief: Prepare a Firmware for the impersonation of a shared Library as defined in [VFF]. This command is optional.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM, that has the same Firmware Identifier as the provided firmware_identifier.

- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- Load the Memory Partitions of the Firmware
- Initiate the instantiation of a Kernel Object for the impersonation
- Initialize the above Kernel Object with the physical memory address of the sub-Memory Partition related to a given Library.
- If error while impersonation
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '09' (uint8_t) MGT_Open_Library_Impersonation command.
- firmware_identifier (UUID_t) identifier of the Firmware (i.e., m_xName in [VFF]).
- index (MK_Index_t) index of the Library Descriptor index within the array of Library descriptors of the Firmware header defined in [VFF]

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3.4 MGT_Close_Library_Impersonation

Brief: Prepare a Firmware for the closing of a library impersonation. This command is optional.

Description:

The command performs the following operations:

- Instruct the kernel to clear the reference related to the sub-partition for a Library to impersonate
- If error while closing impersonation
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '0A' (uint8_t) MGT_Close_Library_Impersonation command.

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3.5 MGT_Open_LLOS_Impersonation

Brief: Prepare the LLOS software for the impersonation of LLOS Software as defined in [VFF]. This command is optional.

Description:

The command performs the following operations:

- Load the Memory Partition of the software related to the LLOS

- If error while impersonating
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '0B' (uint8_t) MGT_Open_LLOS_Impersonation command.
- firmware_identifier (UUID_t) identifier of the software (i.e., m_xName in [VFF]).

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3.6 MGT_Close_LLOS_Impersonation

Brief: Prepare the LLOS software for the closing of a LLOS impersonation. This command is optional.

Description:

The command performs the following operations:

- Instruct the kernel to clear the reference related to the sub-partition for the LLOS to impersonate
- If error while closing impersonation
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

- '0C' (uint8_t) MGT_Close_LLOS_Impersonation command.

Return:

- response_code (uint8_t) management service response code as described in Table 5-5

5.10.3.7 MGT_Allocate_Firmware

Brief: Prepare a Firmware for the impersonation.

Description:

The command performs the following operations:

- If firmware_identifier is not 16 bytes in size
  - Return response_code = MGT_ERROR_ILLEGAL_PARAMETER
- Load the Firmware Header from the MGT Process NVM, that has the same Firmware Identifier as the provided firmware_identifier.
- If Firmware Header is not found
  - Return response_code = MGT_ERROR_UNKNOWN_UUID
- Load the Firmware header from the MGT Process based on provided its identifier
- Allocate memory for Firmware according to the Process Descriptors of the Firmware Header

- Allocated memory is uninitialized. Therefore, the System VPP Application shall fully write into each impersonated Virtual Address Space Region. First with all content of the each provided Firmware Sub Memory Partition, and then pad the rest with zeros (‘00’).
- If error while allocating
  - Return response_code = MGT_ERROR_INTERNAL
- Return response_code = MGT_ERROR_NONE

Parameters:

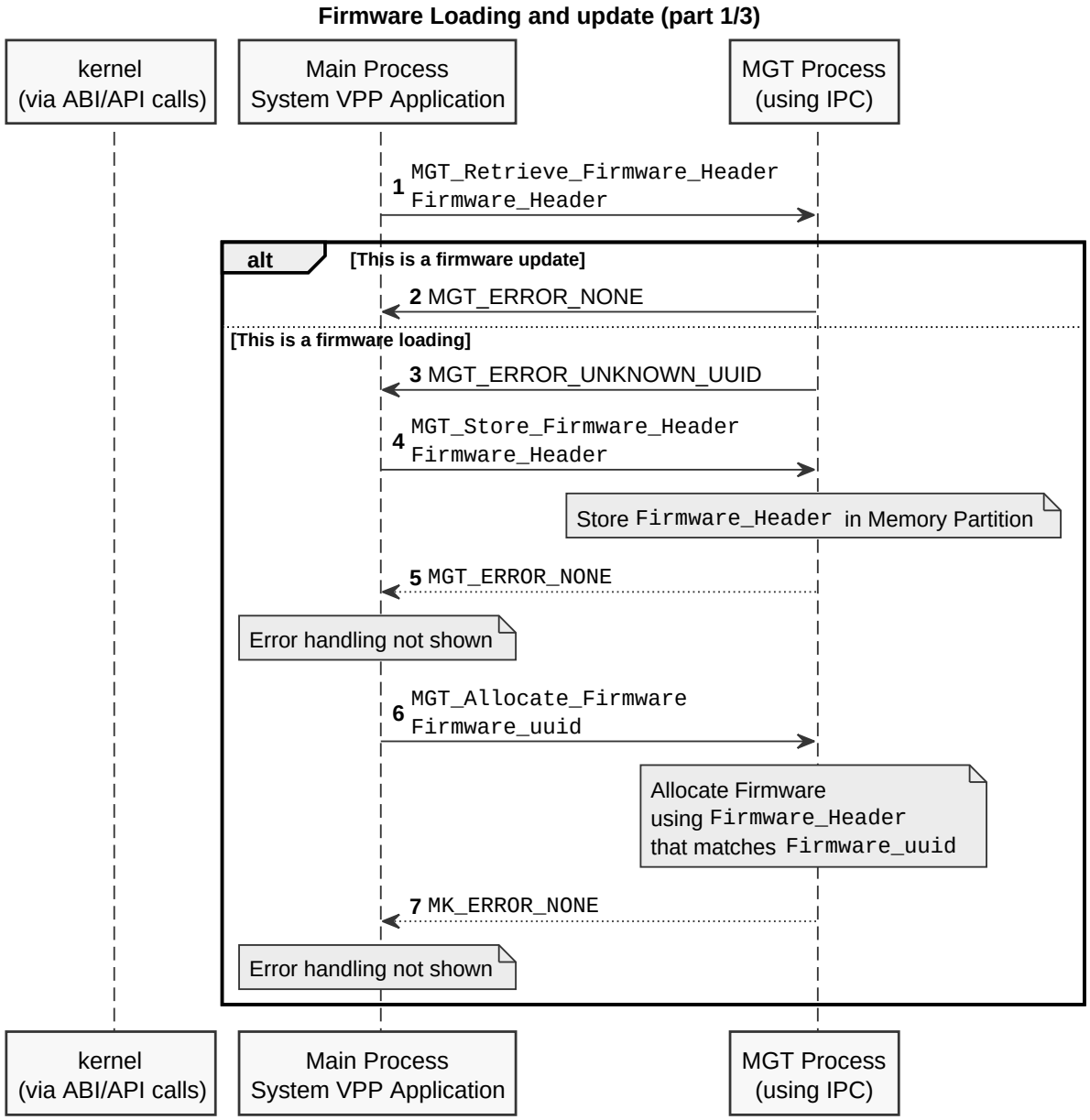
- ‘02’ (uint8_t) MGT_Allocate_Firmware command.
- firmware_identifier (UUID_t] identifier of the Firmware (i.e., m_xName in [VFF]).

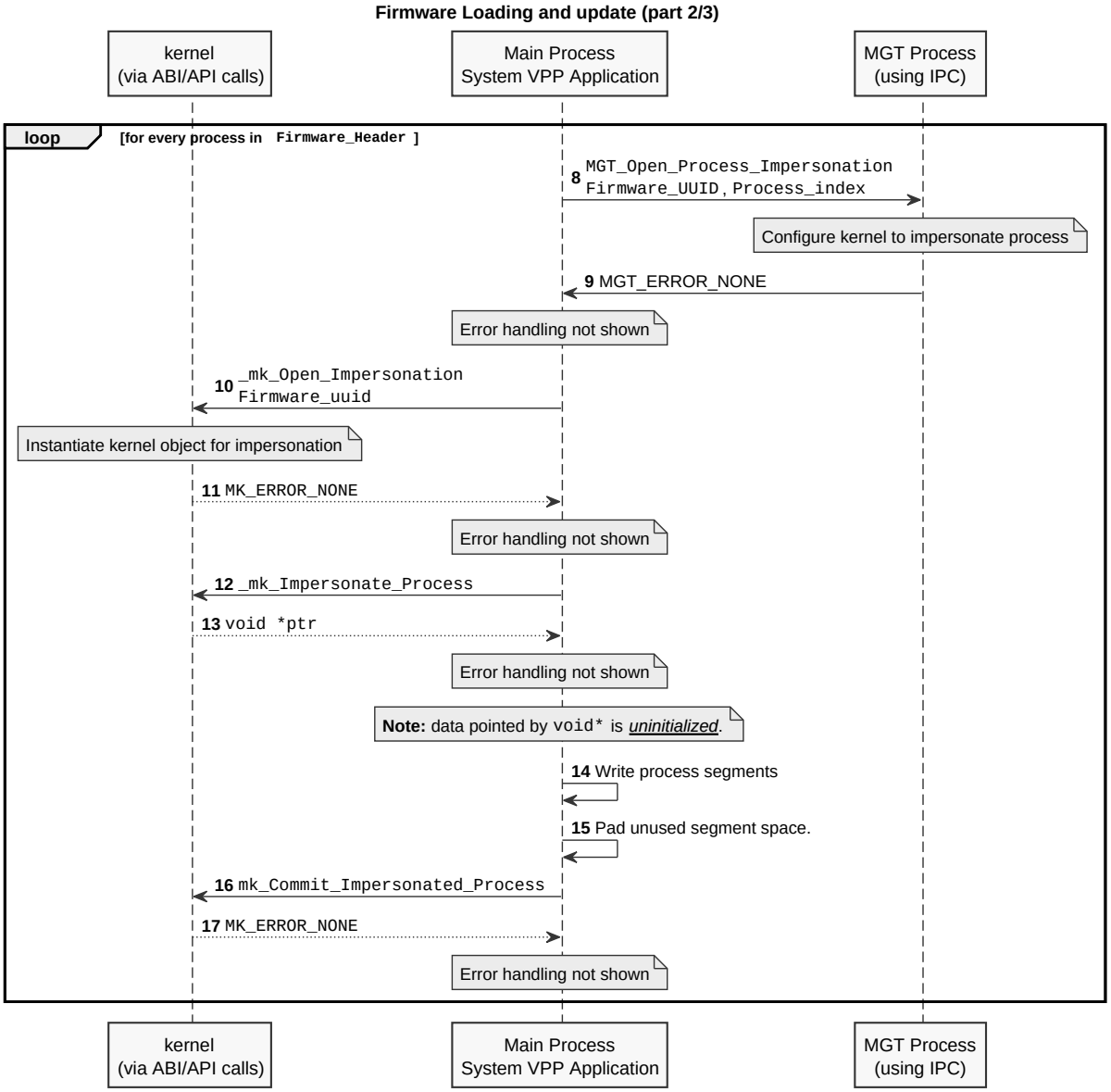
Return:

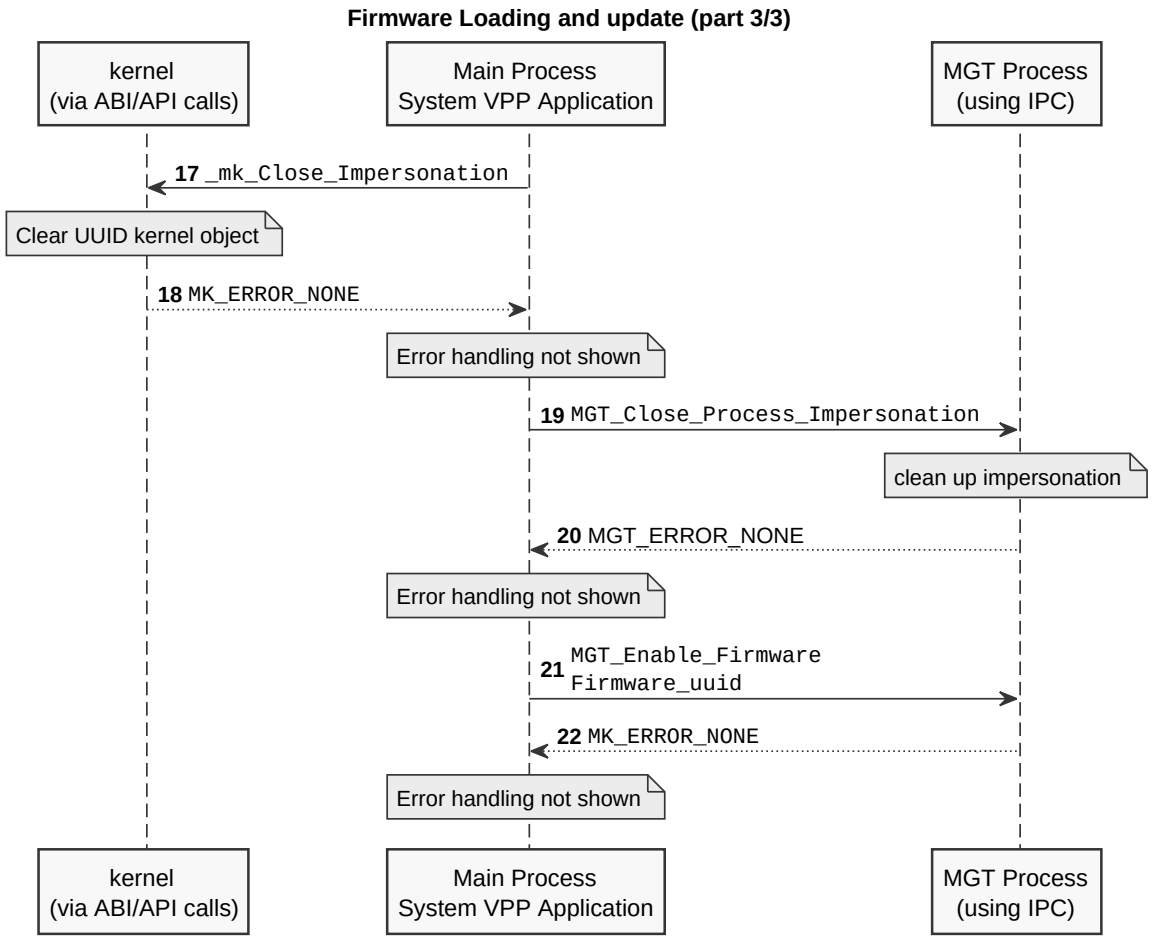
- response_code (uint8_t) management service response code as described in Table 5-5

Figure **5-9** illustrates the installation of a Firmware.

Figure 5-9: Firmware Installation or Update







5.11 Mandatory Access Control

The Mandatory Access Control (MAC) allows controlling the access to:

- Cross-Execution-Domain Mailboxes and IPC defined in section 7.4,
- Group of kernel functions defined in 5.11.3.

Any VPP Application violating the predefined accesses generates a severe Exception (i.e., MK_EXCEPTION_SEVERE).

Cross-Execution-Domain Mailboxes and IPC consider only the following Processes:

- Main Process of a VPP Application,
- COM Process of VPP,
- MGT Process of VPP.

Two types of VPP Application are considered:

- VPP Application
- System VPP Application, having extended rights

5.11.1 VPP Application

Table 5-7: Standard Mandatory Access Control Rules

		FROM					
		COM Process		MGT Process		Main Process	
		IPC	MAILBOX	IPC	MAILBOX	IPC	MAILBOX
TO Process	COM					MK_IPC_MAIN_COM_ID	MK_MAILBOX_MAIN_COM_ID
	MGT						MK_MAILBOX_MAIN_MGT_ID
	Main	MK_IPC_COM_MAIN_ID	MK_MAILBOX_COM_MAIN_ID		MK_MAILBOX_MGT_MAIN_ID		

The VPP Application shall be able to access the following VREs [SREQ**115**]:

- MK_VRE_RNG Random Number Generation hardware function

The VPP Application may access the following VRE:

- MK_VRE_ECC ECC accelerator hardware function
- MK_VRE_RSA RSA accelerator hardware function
- MK_VRE_AES AES accelerator hardware function
- MK_VRE_HASH HASH accelerator hardware function
- MK_VRE_RAF Remote Audit Function

Note: The Primary Platform Maker may provide additional, platform-specific VREs, to be accessible by VPP Application.

5.11.2 System VPP Application

Table 5-8: Access to Cross-Execution-Domain Mailboxes and IPC

		FROM					
		COM Process		MGT Process		Main Process	
		IPC	Mailbox	IPC	Mailbox	IPC	Mailbox
TO Process	COM					MK_IPC_MAIN_COM_ID	MK_MAILBOX_MAIN_COM_ID
	MGT					MK_IPC_MAIN_MGT_ID	MK_MAILBOX_MAIN_MGT_ID
	Main	MK_IPC_COM_MAIN_ID	MK_MAILBOX_COM_MAIN_ID	MK_IPC_MGT_MAIN_ID	MK_MAILBOX_MGT_MAIN_ID		

The System VPP Application shall be able to access the following VRE [SREQ**116**]:

- MK_VRE_RNG Random Number Generation hardware function

The System VPP Application may optionally access the following VREs:

- MK_VRE_ECC ECC accelerator hardware function
- MK_VRE_RSA RSA accelerator hardware function
- MK_VRE_ROT Long-term credentials storage as defined in section 3.1
- MK_VRE_AES AES accelerator hardware function
- MK_VRE_HASH HASH accelerator hardware function
- MK_VRE_RAF Remote Audit Function
- Any VRE reserved by the Primary Platform Makers

5.11.3 Kernel Functions Groups

Table 5-9 defines the groups of kernel functions.

Table 5-9: Groups of Kernel Functions

Groups	Description	Allowed Kernel Functions
MK_MAC_GA	General ABI for any Process	_mk_Get_Error _mk_Get_Exception _mk_Get_Process_Priority _mk_Set_Process_Priority _mk_Suspend_Process _mk_Resume_Process _mk_Request_No_Preemption _mk_Commit

		_mk_RollBack _mk_Yield _mk_Get_Mailbox_Handle _mk_Get_IPC_Handle _mk_Get_Access_IPC _mk_Release_Access_IPC _mk_Get_Mailbox_Handle_Activated _mk_Send_Signal _mk_Wait_Signal _mk_Get_Signal _mk_Get_VRE_Handle _mk_Attach_VRE _mk_Get_Access_VRE _mk_Release_Access_VRE _mk_Get_Time _mk_Get_Process_Handle
MK_MAC_SYS_APP	System ABI for System VPP Application Processes	_mk_Open_Impersonation _mk_Close_Impersonation _mk_Impersonate_Process _mk_Commit_Impersonated

Use of Kernel Functions:

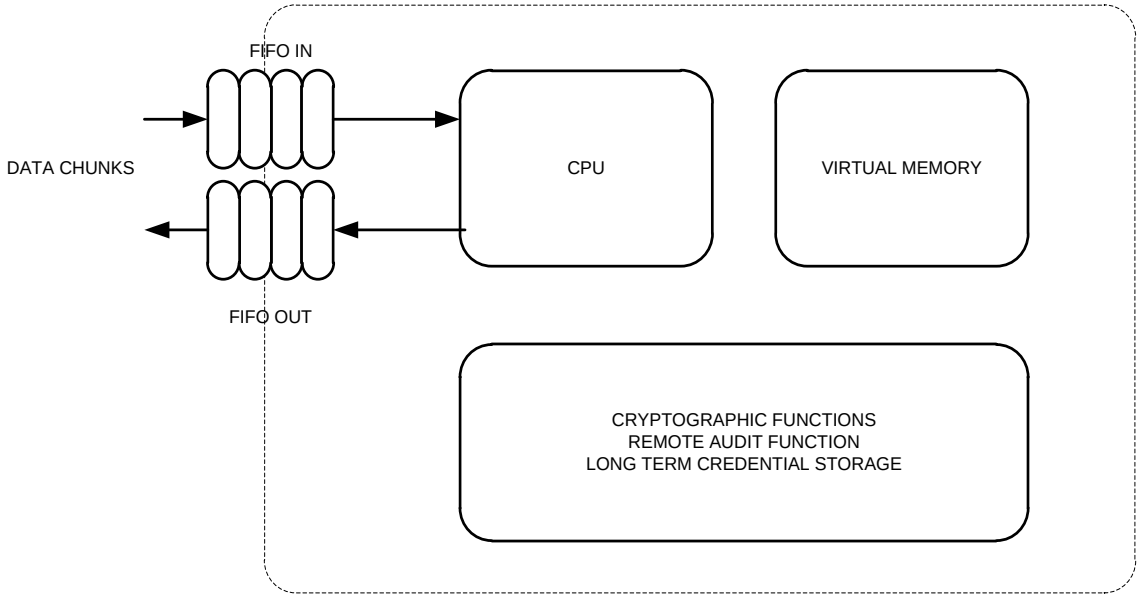
- MK_MAC_GA - is available to all Applications, including VPP Processes
- MK_MAC_SYS_APP – is restricted to System VPP Applications

6 Virtual Primary Platform Application

6.1 The Virtual Hardware Platform

Figure 6-1 illustrates the virtual hardware platform on which the VPP Application is built.

Figure 6-1: Virtual Hardware Platform



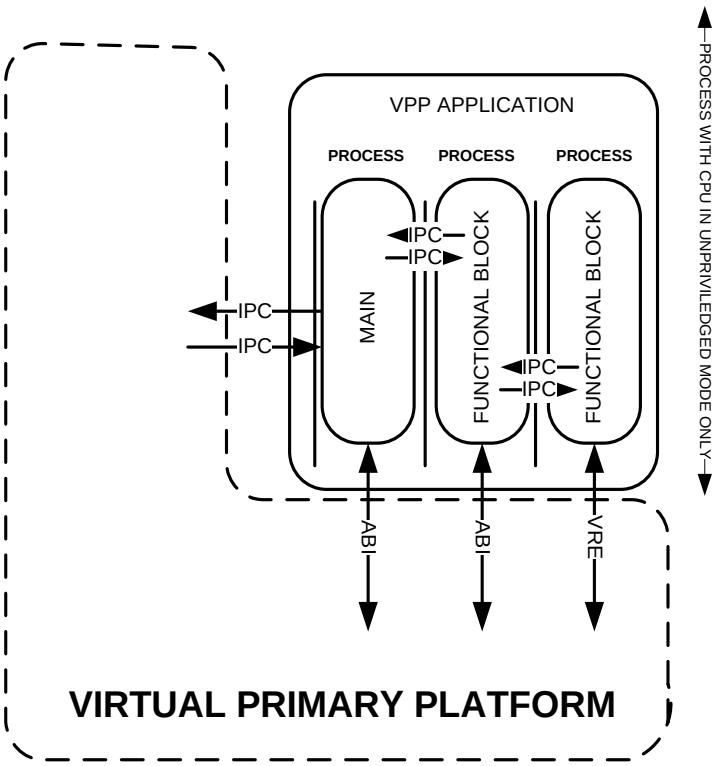
Each Process of the VPP Application runs on top of a virtual hardware platform.

Only the Main Process of the VPP Application shall access a FIFO of data chunks [SREQ**117**].

6.2 Structure

Figure 6-2 illustrates the structure of the VPP Application.

Figure 6-2 : Structure of the VPP Application



The VPP Application may be a collection of Processes. The VPP Application shall run at least a Process named Main [SREQ**118**].

6.3 VPP Application Session

A VPP Application session represents the period of time from VPP Application instantiation to its termination. The following operations are performed during the Firmware instantiation [SREQ**119**]:

- All VPP Application Processes are instantiated:
  - The content of CODE, CONSTANTS, DATA, and the optional LIB_CODE and LIB_CONSTANTS of each Process is initialized with the data previously written by the Firmware Loader.
  - The NVM of each Process is initialized by the content of the last successful _mk_Commit operation or by initial values provided by the Firmware Loader.
  - The Virtual/Physical Memory[15] Space content of each STACK is zeroed.
  - The Virtual Address Space content of each Writer IPC in is zeroed, including the Cross-Execution Domain IPCs.
  - The Writer IPCs are cleared.
  - All Processes are instantiated in the “Suspended R” state with the default priority *MK_PROCESS_PRIORITY_NORMAL.* The entry point address for each Process is provided in the Firmware, as defined in [VFF].

[15] The stack of the process may be mapped to the Virtual Address Space or to the Physical Address Space (Primary Platform implementation dependent).

- The VPP COM Process shall [SREQ**120**]:
  - Reset the m_Read$_{IN}$ field of its FIFO OUT.
  - Write the incoming data chunks, if any, in its FIFO OUT.
  - Notify the Main Process that its FIFO IN has been updated
- The MGT Process resumes the Main Process.
- The Main Process copies:
  - The values m_MTU_IN and m_Size_IN from its FIFO IN within the IPC referenced by MK_IPC_COM_MAIN_ID,
  - The values in m_MTU_OUT and m_Size_OUT to its FIFO OUT within the IPC referenced by MK_IPC_MAIN_COM_ID.

A VPP Application may restart itself (and therefore its session) by sending the Signal MK_SIGNAL_APP_RESTART to the Mailbox identified as MK_MAILBOX_MAIN_MGT_ID in the MGT Process.

A VPP Application session shall end when VPP Application termination begins [REQ**121**].

A VPP Application termination shall begin [SREQ**122**]:

- When the Mailbox identified as MK_MAILBOX_MGT_MAIN_ID in the Main Process receives the MK_SIGNAL_KILL_REQUESTED
- Or when the Mailbox identified as MK_MAILBOX_MAIN_MGT_ID in the MGT Process receives the MK_SIGNAL_KILL_ITSELF

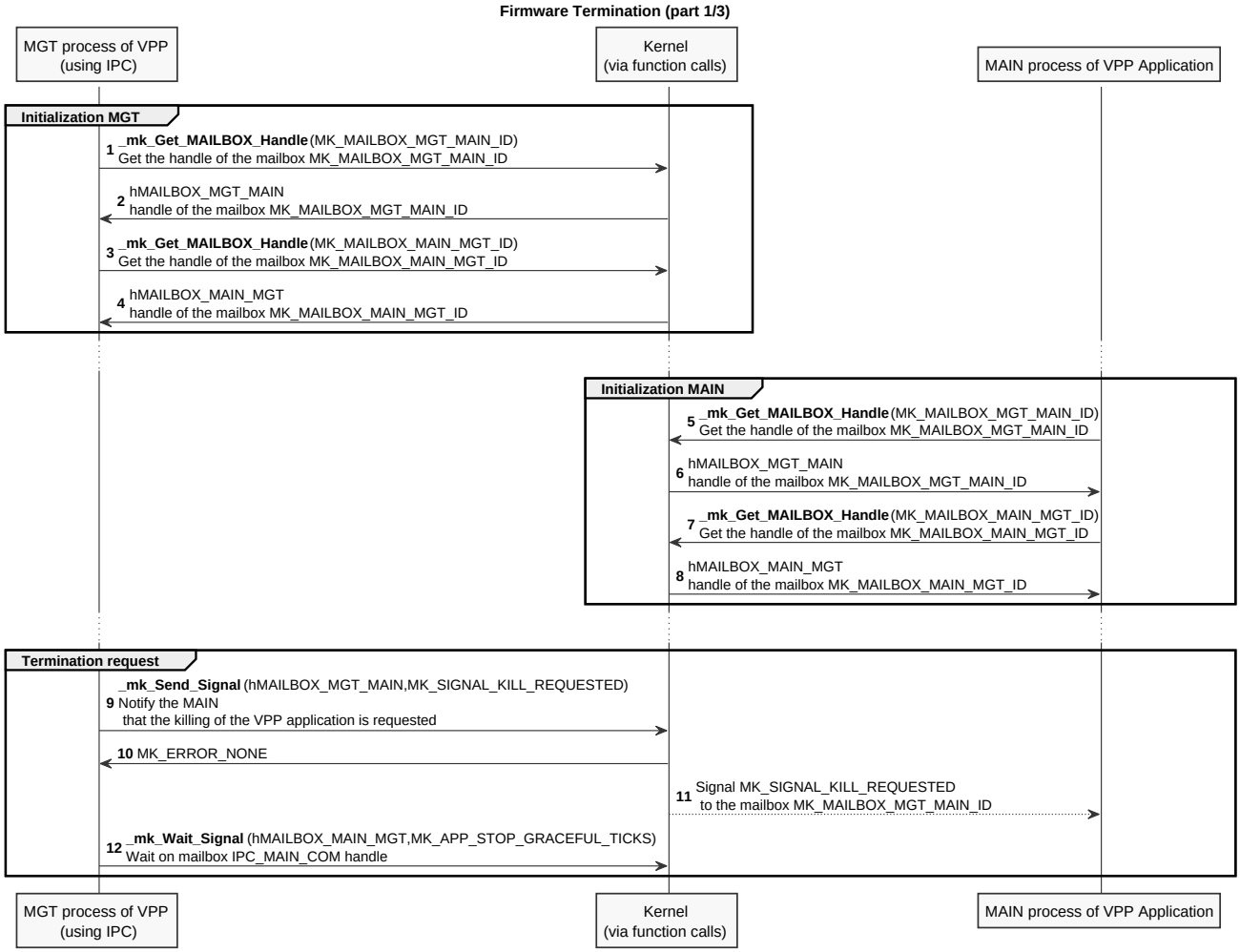
VPP shall complete VPP Application termination [REQ**123**]:

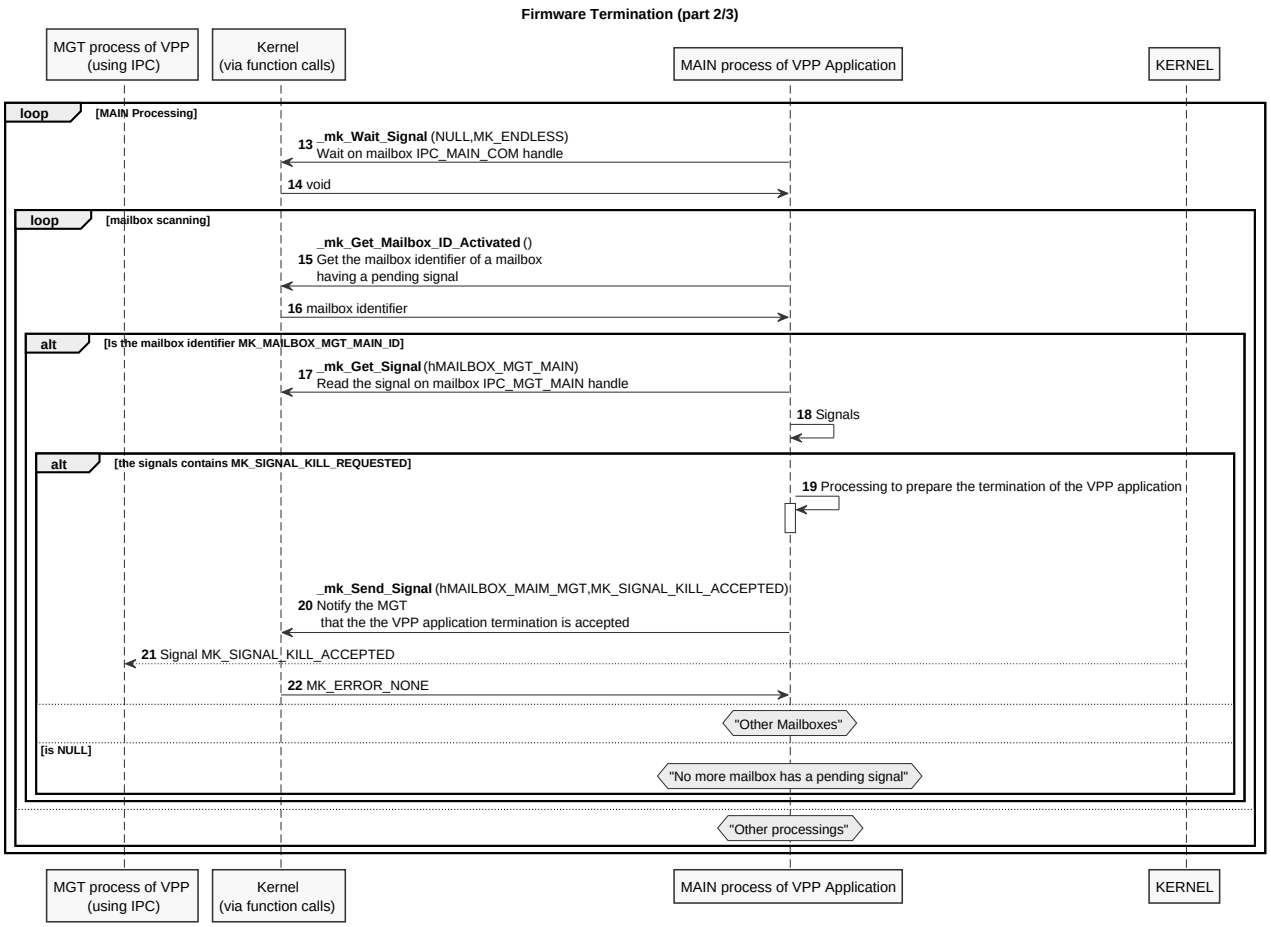
- When the Mailbox identified as MK_MAILBOX_MAIN_MGT_ID in the MGT Process receives the MK_SIGNAL_KILL_ACCEPTED or MK_SIGNAL_KILL_ITSELF
- Or after VPP Application MK_APP_STOP_GRACEFUL_TICKS time elapsed [SREQ**124**] The following applies when the VPP Application terminates [SREQ**125**]:
  - All data chunks in the FIFO OUT of the COM and Main Processes which have not been read are lost.
  - Only changes committed to NVM Memory Partitions are preserved.

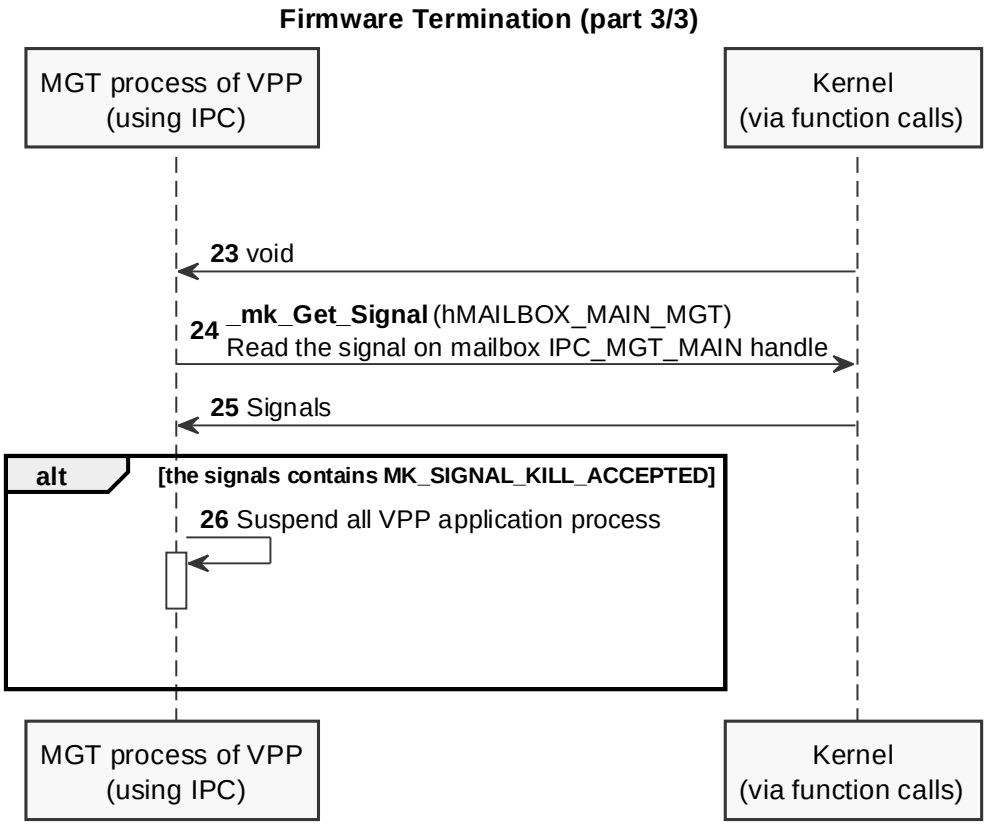
VPP shall ensure that VPP Application restart does not override VPP Application termination [SREQ**126**].

Figure 6-3 illustrates the termination of the VPP Application.

Figure 6-3: VPP Application Termination







6.4 High Level Operating System (HLOS)

The HLOS may support one or several Applications [REQ**127**].

The HLOS may claim conformance to [HLOS02], [HLOS34], [HLOS25], [HLOS14], [HLOS42] and [HLOS93].

6.4.1 HLOS Application

An application running on a HLOS supporting a suitable configuration based on [HLOS02], [HLOS34], [HLOS25], [HLOS14], [HLOS42] and [HLOS93] may claim conformance with [102 221], [103 383], [102 223], [7816-4], [103 384] and [102 622].

6.4.2 Remote Application Management

The HLOS may support Remote Application Management of non-native applications running on an application framework (e.g. Java Card) of the HLOS.

7 Minimum Level of Interoperability (MLOI)

Note: Interoperability is the capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units [2832].

Note: Some constants and data types defined in this section are used in other documents defining VPP, e.g. [VFF].

7.1 Basic Data Types

The Primary Platform shall at least be based on 32-bit architecture CPU [REQ**128**]. The endianness is Primary Platform Dependent.

Note: Through Table 7-1 XLEN is the size in bytes memory address. It is a platform dependent value and for 32bit platforms XLEN equals 4 bytes, and 8 bytes for 64bit platforms.

Table 7-1 defines the basic data types used within the Firmware header.

Table 7-1: Basic Data Types

Type	Description	Size (byte)
uint8_t	8-bit unsigned integer	1
uint16_t	16-bit unsigned integer	2
uint32_t	32-bit unsigned integer	4
uint64_t	64-bit unsigned integer	8
VPP_FRW_TYPE_e	Enumerated VPP firmware/software type as defined in Table 7-6	1
MK_Index_t	Index of an element in a typed array	2
MK_IPC_ID_u	Composite Identifier of an IPC as defined in Table 7-2	2
MK_MAILBOX_ID_u	Composite Identifier of a Mailbox as defined in Table 7-2	2
MK_PROCESS_ID_u	Composite Identifier of a Process as defined in Table 7-2	2
MK_PROCESS_PRIORITY_e	Priority of a Process as defined as defined in Table 7-4	2
MK_LIB_ID_u	Composite Identifier of a shared library	2
MK_VRE_e	Enumerated VRE Identifier	4
PPROCESS_Function_t	Memory address to a Process entry point	XLEN
PLLOS_Function_t	Memory address to a LLOS entry point	XLEN
StackType_t	Stack element	XLEN
UUID_t	Unique Universal IDentifier	16

v32_u	void 32-bit	4
MK_ERROR_e	Enumerated type for errors	2
MK_EXCEPTION_e	Enumerated type for Exceptions	2
MK_BITMAP_t	32-bit bitmap for Exception, Signal or LIB Descriptor conveyor	4
MK_SIGNAL_e	Enumerated Signal type as defined in Table 7-8	4
MK_HANDLE_t	Handle to a Kernel Object	XLEN
MK_TIME_t	Time unsigned 64-bit integer (uint64_t)	8

Table 7-2: Composite Type Identifiers

Composite Type Identifier	**Execution Domain type (unsigned integer bit field) Bit [15-14]**	**Enumerated Identifier (unsigned integer bit field) Bit [13-0]**	**Description**
MK_IPC_ID_u	MK_DOMAIN_TYPE_e as defined in Table 7-3	MK_IPC_ID_e	Identifier of an IPC
MK_MAILBOX_ID_u		MK_MAILBOX_ID_e	Identifier of a Mailbox
MK_PROCESS_ID_u		MK_PROCESS_ID_e	Identifier of a Process
MK_LIB_ID_u		MK_LIB_ID_e	Identifier of a shared library

Table 7-3: Execution Domain Types MK_DOMAIN_TYPE_e

Type	**Enumerated Identifier Bit [1-0]**	**Domain**	**Value**
MK_EXECUTION_DOMAIN_TYPE_e	MK_EXECUTION_DOMAIN_TYPE_VPP	System VPP Execution Domain	b10
	MK_EXECUTION_DOMAIN_TYPE_APP	VPP Application Execution domain	b01

Table 7-4: Priority values of a Process

Priority	Definition	Value
MK_PROCESS_PRIORITY_LOW	Lowest priority	‘0000’
MK_PROCESS_PRIORITY_NORMAL	Normal Priority (Default)	‘0004’
MK_PROCESS_PRIORITY_HIGH	Highest priority	‘0008’
MK_PROCESS_PRIORITY_ERROR	Indicates error when Process priority is retrieved	‘FFFF’

Table 7-5 defines the interoperable identifiers for accessing hardware resources[16].

Table 7-5: VRE Identifiers

Identifier	Definition	Value
MK_VRE_AES	Access to the interface of the AES function	‘01’
MK_VRE_ECC	Access to the interface of the ECC function	‘04’
MK_VRE_RSA	Access to the interface of the RSA function	‘08’
MK_VRE_ROT	Access to the interface of the Long-term credentials storage	‘10’
MK_VRE_HASH	Access to the interface of the Hash function	‘20’
MK_VRE_RNG	Access to the interface of the RNG function	‘40’
MK_VRE_RAF	Access to the interface of the Remote Audit Function	‘80’
MK_VRE_ DOMAIN_BASE	Access to the interfaces of additional Execution Domain hardware functions	‘100’

Note: This table enumerates VRE Identifiers for different and possibly optional hardware functions.

Table 7-6 defines the different types of executable code.

[16] VRE usage is hardware/implementation dependent. For example, two Primary Platform implementations may use different RNG hardware and thus require different handling of the MK_VRE_RNG

Table 7-6: Firmware/Software Types

Identifier	Definition	Value
FIRMWARE_SOFTWARE_TYPE_APP	The Firmware of a VPP Application	‘01’
FIRMWARE_SOFTWARE_TYPE_VPP	The Primary Platform Software excluding the LLOS software	‘02’
FIRMWARE_SOFTWARE_TYPE_SYSAPP	The Firmware of the System VPP Application	‘04’
FIRMWARE_SOFTWARE_TYPE_LLOS	The software of the LLOS	‘08’

Table 7-7: Scheduling Types

Identifier	Definition	Value
MK_SCHEDULING_TYPE_COLLABORATIVE	Collaborative scheduling	‘01’
MK_SCHEDULING_TYPE_PREEMPTIVE	Pre-emptive scheduling	‘02’

Table 7-8: Signal Identifiers

MK_SIGNAL_ID_e	Description	Value
MK_SIGNAL_TIME_OUT	Timeout notification	‘00000001’
MK_SIGNAL_ERROR	_mk_Get_Signal function or VRE has generated an error	‘00000002’
MK_SIGNAL_EXCEPTION	Notification for an Exception from a child Process	‘00000004’
MK_SIGNAL_DOMAIN_BASE_0	Mailbox defined Signals for generic use within the scope of the Execution Domains MK_EXECUTION_DOMAIN_TYPE_VPP and MK_EXECUTION_DOMAIN_TYPE_APP as defined in Table 7-3	‘00000008’
MK_SIGNAL_DOMAIN_BASE_1		‘00000010’
MK_SIGNAL_DOMAIN_BASE_2		‘00000020’
MK_SIGNAL_DOMAIN_BASE_3		‘00000040’
MK_SIGNAL_DOMAIN_BASE_4		‘00000080’
MK_SIGNAL_DOMAIN_BASE_5		‘00000100’
MK_SIGNAL_DOMAIN_BASE_6		‘00000200’
MK_SIGNAL_DOMAIN_BASE_7		‘00000400’
MK_SIGNAL_DOMAIN_BASE_8		‘00000800’
MK_SIGNAL_DOMAIN_BASE_9		‘00001000’
MK_SIGNAL_DOMAIN_BASE_10		‘00002000’
MK_SIGNAL_DOMAIN_BASE_11		‘00004000’
MK_SIGNAL_DOMAIN_BASE_12		‘00008000’
MK_SIGNAL_DOMAIN_BASE_13		‘00010000’
MK_SIGNAL_DOMAIN_BASE_14		‘00020000’
MK_SIGNAL_DOMAIN_BASE_15		‘00040000’
MK_SIGNAL_DOMAIN_BASE_16		‘00080000’
MK_SIGNAL_DOMAIN_BASE_17		‘00100000’
MK_SIGNAL_DOMAIN_BASE_18		‘00200000’
MK_SIGNAL_DOMAIN_BASE_19		‘00400000’
MK_SIGNAL_DOMAIN_BASE_20		‘00800000’
MK_SIGNAL_DOMAIN_BASE_21		‘01000000’
MK_SIGNAL_DOMAIN_BASE_22		‘02000000’
MK_SIGNAL_DOMAIN_BASE_23		‘04000000’
MK_SIGNAL_DOMAIN_BASE_24		‘08000000’
MK_SIGNAL_DOMAIN_BASE_25		‘10000000’
MK_SIGNAL_DOMAIN_BASE_26		‘20000000’
MK_SIGNAL_DOMAIN_BASE_27		‘40000000’

MK_SIGNAL_DOMAIN_BASE_28		'80000000'

7.2 Constants and Limits

Table 7-9: defines the constants and limits related to the Firmware format or a Firmware or the VPP Application as the runtime instance of the Firmware. Other limits may apply to the Primary Platform.

The Primary Platform shall support the Constants and Limits as described in Table 7-9 [REQ**129**].

Primary Platform-dependent values, as described in Table 7-10, shall be provided by the Primary Platform Maker [REQ**130**].

Table 7-9: Constants and Limits for any Primary Platform

Name	Description	Value
MK_APP_STOP _GRACEFUL_TICKS	A grace period, in ticks, given to a VPP Application, so it may shut down gracefully	10
MK_IPC_ DOMAIN_BASE _ID	Minimal enumerated IPC identifier within the scope of an Execution Domain (including BASE_ID)	'100'
MK_IPC_COM_LENGTH	Length of the IPC identified by MK_IPC_MAIN_COM_ID and MK_IPC_COM_MAIN_ID	4KB
MK_IPC_LIMIT	Maximal number of IPC descriptors per Firmware	64
MK_IPC_MAX_ID	Maximal enumerated IPC identifier value within the scope of an Execution Domain (including MAX_ID)	'3FFF'
MK_IPC_MGT_LENGTH	Length of the IPC identified by MK_IPC_MAIN_MGT_ID and MK_IPC_MGT_MAIN_ID	6KB
MK_IPC_SIZE_LIMIT	Maximal IPC length	32KB
MK_LIB_ DOMAIN_BASE _ID	Minimal enumerated library identifier within the scope of an Execution Domain (including BASE_ID)	'100'
MK_LIB_LIMIT	Maximal number of LIB Descriptors in the Firmware	32
MK_LIB_MAX_ID	Maximal enumerated shared library identifier value within the scope of an Execution Domain (including MAX_ID)	'3FFF'
MK_MAILBOX_DOMAIN _BASE_ID	Minimal enumerated Mailbox identifiers within the scope of an Execution Domain	'100'
MK_MAILBOX_LIMIT	Maximal number of Mailbox descriptors per Firmware excluding the kernel Mailbox	64
MK_MAILBOX_MAX_ID	Maximal enumerated Mailbox identifier value within the scope of a domain (including MAX_ID)	'3FFF'
MK_MIN _CONCURRENT_IPC_LIMIT	Minimal number of IPCs accessible concurrently by a Process	6
MK_MIN_STACKS_SUM	The minimal size in bytes, supported by the Primary Platform, for the sum of all stack memory used by	24K

_SUPPORTED	all Processes in a VPP Application	
MK_MIN_APP_IPC	Minimal number of IPC descriptors in a Firmware	0
MK_MIN_APP_MAILBOXES	Minimal number of Mailbox descriptor of a Firmware, not including kernel Mailboxes.	0
MK_MIN_APP_PROCESSS	Minimal number of Process supported by a VPP Application.	1
MK_MIN_SUPPORTED _MEMORY_PARTITION_SIZE	Minimal size in bytes of Memory Partition in [VFF] supported by the Primary Platform	8MB
MK_MIN_VIRTUAL _MEMORY_SIZE[17]	Minimum size of the Virtual Memory that the MMF shall manage	1KB
MK_PROCESS _ DOMAIN_BASE _ID	Minimal enumerated Process identifier within the scope of an Execution Domain (including BASE_ID)	'100'
MK_PROCESS_LIMIT	Maximal number of Process Descriptors in the Firmware	32
MK_MAX_PROCESS_ ID	Maximal enumerated Process identifier value within the scope of an Execution Domain (included)	'3FFF'
MK_MIN_SUPPORTED _STACK	Minimal stack size supported for a Process, given in StackType_t units	512 StackType_t units (2KB if 32bit)

Table 7-10: Primary Platform Dependent Constants and Limits

Name	Description
MK_ MEMORY_PARTITION_SIZE	The size in bytes of Memory Partition. Shall be greater than MK_MIN_SUPPORTED_MEMORY _PARTITION_SIZE
MK_AVERAGE_COMMIT_TIME	Average NVM transaction time (tick unit)
MK_BEGIN_VSPACE	Virtual memory address of the beginning of the Virtual Address Space
MK_IS_LITTLE_ENDIAN	0 – false 1 – true
MK_IS_PREEMPTIVE _SCHEDULING_SUPPORTED	Define the type of scheduling being supported for VPP Applications 0 – false 1 - true
MK_MAX _STACKS_SUM_SUPPORTED	The maximal size in bytes, supported by the Primary Platform, for the sum of all stack memory used by all Processes in a VPP

[17] This information allows the VPP Application designer to prevent some software side channel attacks.

	Application.
MK_MAX_CONTEXT _SWITCH_TIME	Maximal time in ticks for switching between two Processes
MK_MAX_RAF_TIME_MS	Maximal time for performing a Remote Audit Function operation in milliseconds
MK_NO_PREEMPTION _MAX_TIMEOUT	Maximal time duration during which the VPP Application Process cannot be pre-empted by a VPP Process (tick unit) 0 if _mk_Request_No_Preemption is not supported
MK_PERFORMANCE _INDEX_METHOD	Performance benchmark method, to be declared by Primary Platform Maker
MK_PERFORMANCE_INDEX	The performance of Primary Platform based on given benchmark method
MK_TICK_PER_MS	Value of the kernel tick time in milliseconds
MK_VSPACE_REGION_SIZE	The size in bytes of the Virtual Address Space Region. Shall be equal or greater than MK_ MEMORY_PARTITION_SIZE.

7.3 Errors and Exceptions

Table 7-11 defines the Exception values.

Table 7-11: Exceptions

Exception Name	Description	Rank Value
MK_EXCEPTION_ERROR	An error has occurred in a child of the Process	0
MK_EXCEPTION_SEVERE	A severe Exception has occurred (e.g. memory violation)	1
MK_EXCEPTION _CHILD_PROCESS_DIED	A child Process has died	2
MK_EXCEPTION_VRE_DETACHED	A VRE has been detached while a Process was waiting on it	3
MK_EXCEPTION_VENDOR_BASE	Starting index for VPP implementation-specific Exceptions	16

MK_EXCEPTION_MAX	Maximal Exception rank value allowed	31

The Exception type is MK_EXCEPTION_e as an enumerated value representing the bit rank (power of 2) within a 32 bit bitmap (MK_BITMAP_t).

Table 7-12 defines the error values.

Table 7-12: Errors

Error Name	Description	LSB Value
MK_ERROR_NONE	No error	0
MK_ERROR_UNKNOWN_UUID	Unknown UUID	1
MK_ERROR_SEVERE	Severe error	2
MK_ERROR_ILLEGAL_PARAMETER	Illegal parameter	3
MK_ERROR_UNKNOWN_ID	Unknown identifier	4
MK_ERROR_UNKNOWN_HANDLE	Unknown Handle	5
MK_ERROR_UNKNOWN_PRIORITY	Unknown priority	6
MK_ERROR_ACCESS_DENIED	Access denied	7
MK_ERROR_INTERNAL	Internal error	8
MK_ERROR_VENDOR_BASE	Reserved for VPP implementation-specific	32
MK_ERROR_MAX	Maximal error value	255

The Exception type is MK_ERROR_e (16 bit) where the eighth most significant bits (MSBs) are the complementary bits of the eighth least significant bits (LSBs).

7.4 Cross-Execution-Domain Identifiers

Table 7-13 defines the Cross-Execution-Domain Composite Identifiers.

Table 7-13: Cross-Execution Domain Composite Identifier

Identifiers	Domain type Bit [15-14]	Enumerated Identifier Bit [13-0]	Description
MK_PROCESS_COM_VPP_ID	MK_EXECUTION_DOMAIN_TYPE_VPP	0	VPP COM Process

MK_PROCESS_MGT_VPP_ID	MK_EXECUTION_DOMAIN_TYPE_ VPP	1	MGT Process
MK_PROCESS_MAIN_APP_ID	MK_EXECUTION_DOMAIN_TYPE_APP	0	Main Process
MK_MAILBOX_COM_MAIN_ID	MK_EXECUTION_DOMAIN_TYPE_VPP	0	COM Process Mailbox (sender: Main Process)
MK_MAILBOX_MGT_MAIN_ID	MK_EXECUTION_DOMAIN_TYPE_VPP	1	MGT Process Mailbox (sender: Main Process)
MK_MAILBOX_MAIN_COM_ID	MK_EXECUTION_DOMAIN_TYPE_APP	0	Main Process Mailbox (sender: COM Process)
MK_MAILBOX_MAIN_MGT_ID	MK_EXECUTION_DOMAIN_TYPE_APP	1	Main Process Mailbox (sender: MGT Process)
MK_IPC_MAIN_COM_ID	MK_EXECUTION_DOMAIN_TYPE_APP	0	IPC from the Main Process to the COM Process
MK_IPC_COM_MAIN_ID	MK_EXECUTION_DOMAIN_TYPE_VPP	0	IPC from the COM Process to the Main Process
MK_IPC_MAIN_MGT_ID[18]	MK_EXECUTION_DOMAIN_TYPE_APP	1	IPC from the Main Process to the MGT Process
MK_IPC_MGT_MAIN_ID[18]	MK_EXECUTION_DOMAIN_TYPE_VPP	1	IPC from the MGT Process to the Main Process

Cross-Execution-Domain IPCs and Mailbox descriptors are automatically instantiated by the kernel. As such, they cannot be defined in by Firmware. Their ID and IPC size are fixed.

7.5 Cross-Execution-Domain Signals

Table 7-14 defines the Cross-Execution-Domain Signals.

Table 7-14: Cross-Execution-Domain Signals

Identifiers	Value	Description
MK_SIGNAL_IPC_UPDATED	MK_SIGNAL_DOMAIN_BASE_0	The IPC updated
MK_SIGNAL_KILL_REQUESTED	MK_SIGNAL_DOMAIN_BASE_1	MGT signaled the Main Process to terminate itself
MK_SIGNAL_KILL_ACCEPTED	MK_SIGNAL_DOMAIN_BASE_1	The Main Process signaled MGT that it has accepted the kill request
MK_SIGNAL_APP_RESTART	MK_SIGNAL_DOMAIN_BASE_2	The Main Process signaled MGT to restart the VPP Application
MK_SIGNAL_KILL_ITSELF	MK_SIGNAL_DOMAIN_BASE_3	The Main Process committed suicide

[18] The IPCs between MAIN and MGT Processes are valid for the system VPP Application only.

As a Mailbox has only a single reader and writer, Cross-Execution-Domain Signals use the same values when different Signals are used by different Mailboxes. For example, MK_SIGNAL_KILL_REQUESTED has the same value as MK_SIGNAL_KILL_ACCEPTED, but they are being used on different mailboxes.

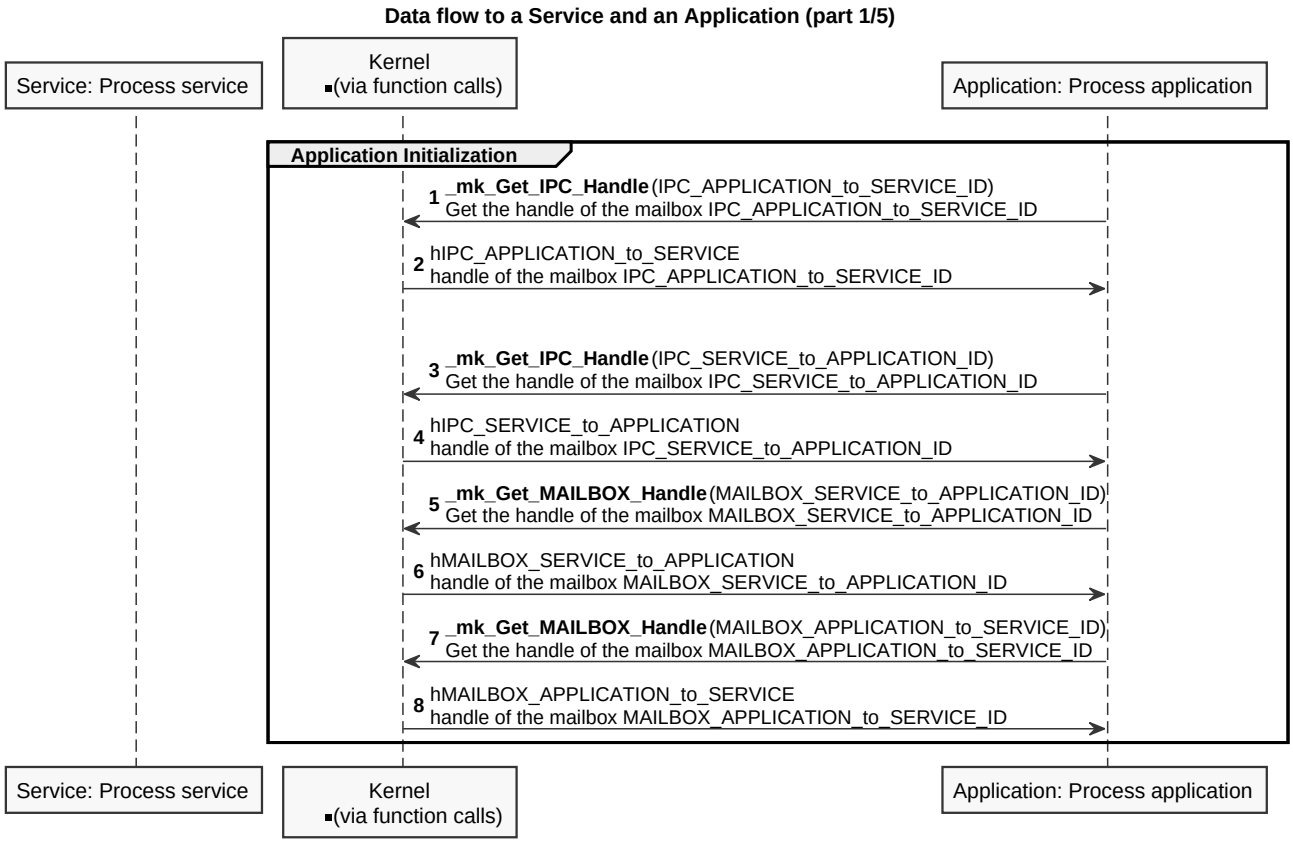
8 Annexes

8.1 Services

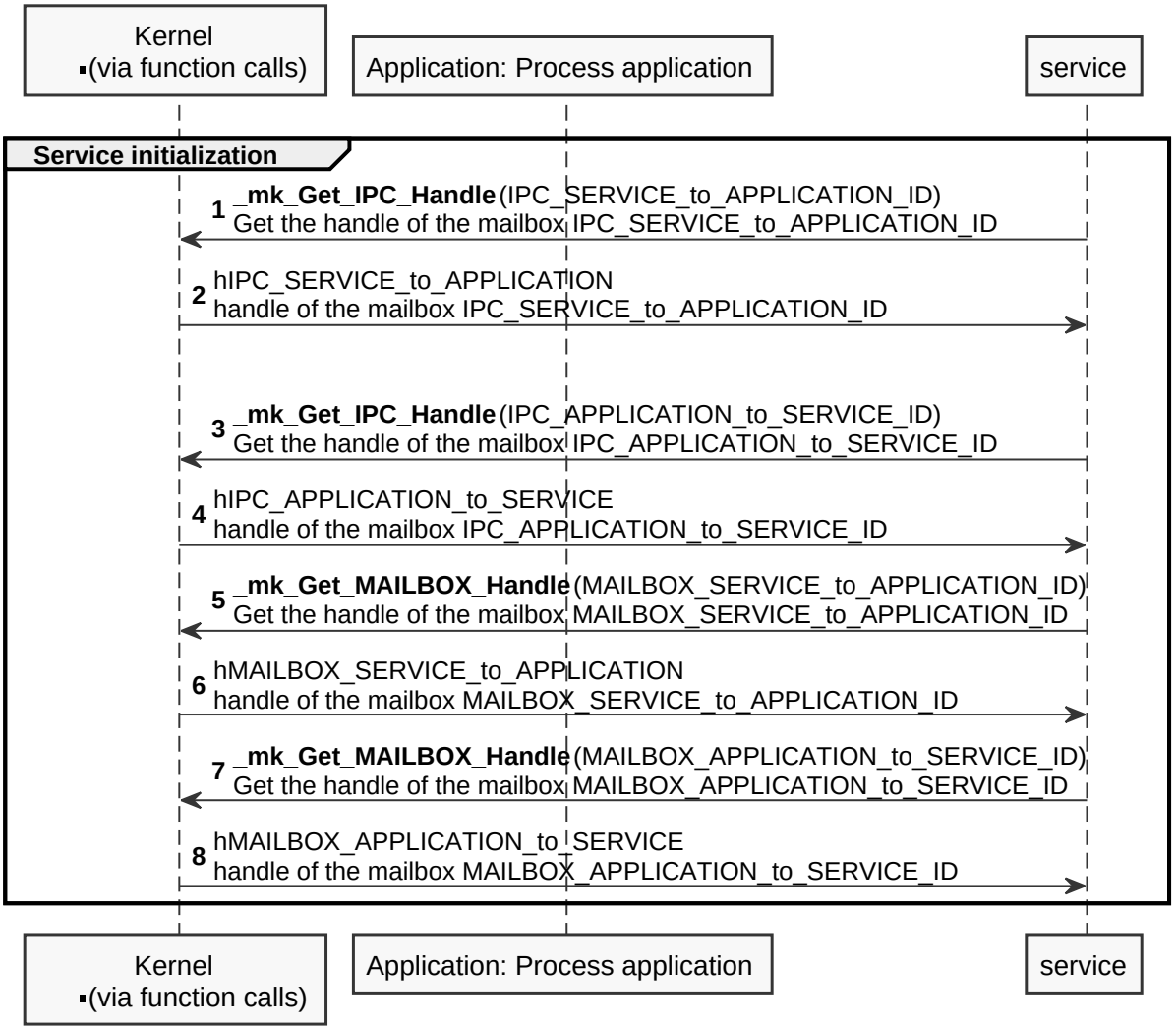
8.1.1 Service Generic Message Flow

Figure 8-1 illustrates a generic data flow between a service providing a function and a Process consuming that function.

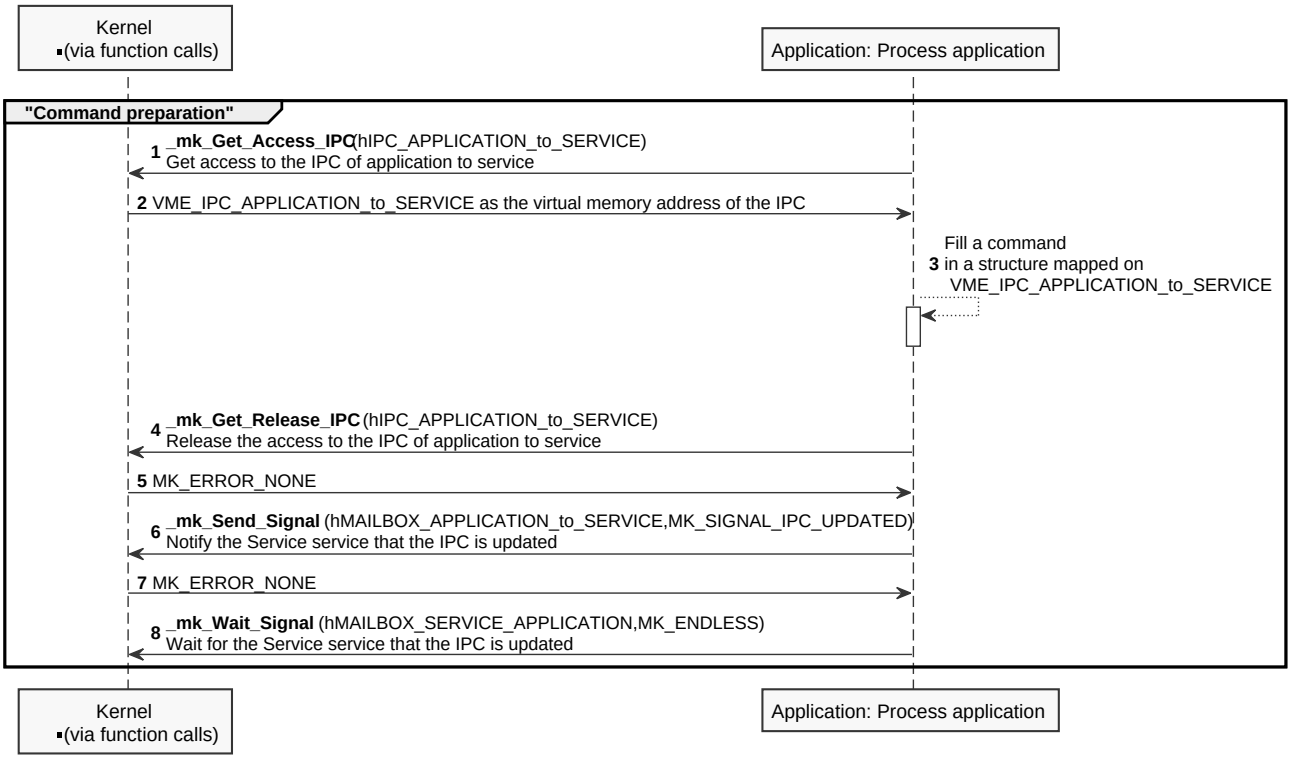
Figure 8-1: Service Generic Messages Flow



Data flow to a Service and an Application (part 2/5)



Data flow to a Service and an Application (part 3/5)



Data flow to a Service and an Application (part 4/5)