Security Target for L4Re Secure Separation Kernel CC 1.0.2

BSI Administration ID: BSI-DSZ-CC-1177-v2-2026

Version 2.1700

Kernkonzept GmbH
Buchenstr. 16b
01097 Dresden
Germany

Prior Work

This document is derived from the ST for Infodas MOS. This original ST is copyright Infodas GmbH and Kernkonzept GmbH.

Table of contents

1 Introduction....7
1.1 Security Target Identification....7
1.2 TOE Identification....7
1.3 TOE Overview....7
1.3.1 TOE Type....8
1.3.2 Intended Method of Use....8
1.3.3 Major Security Features....9
1.3.4 Product Types....9
1.3.5 Required Hardware and Software....9
1.4 TOE Description....10
1.4.1 System Architecture....10
1.4.2 Major Security Features....13
1.4.3 Communication Proxies....17
1.4.4 TOE Boundaries....20
2 Conformance Claims....22
2.1 Conformance with CC Parts 2 and 3....22
2.2 Conformance with Protection Profiles....22
3 Security Problem Definition....23
3.1 Threats....23
3.1.1 Assets....23
3.1.2 Threat Agents....23
3.1.3 Threats countered by the TOE....23
3.2 Organizational Security Policies....24
3.3 Assumptions....24
4 Security Objectives....25
4.1 Security Objectives for the TOE....25
4.2 Security Objectives for the Operational Environment....25
4.3 Rationale for Security Objectives....25
5 Extended Components Definition....27
6 Security Requirements....28
6.1 Security Functional Requirements....28
6.1.1 User Data Protection (FDP)....28

6.1.2 Security Management (FMT)....38
6.1.3 Identification and Authentication (FIA)....39
6.1.4 Privacy (FPR)....39
6.2 Rationale for Security Functional Requirements....39
6.2.1 Coverage....39
6.2.2 Sufficiency....41
6.2.3 SFR Dependencies....43
6.3 Security Assurance Requirements....45
6.4 Security Assurance Requirements Rationale....46
6.5 Security Assurance Requirements Dependency Analysis....46
7 TOE Summary Specification....47
7.1 Separation of Compartments....47
7.2 Information Flow Control....49
7.3 System Management....50
7.4 SFR to TSS References....50
8 Terms and Definitions....52
9 Abbreviations....59
10 References....61

Index of Tables

Table 1: Assets....23
Table 2: Coverage of Security Objectives for the TOE....26
Table 3: Coverage of Security Objectives for the TOE environment....26
Table 4: Objects....33
Table 5: Mapping of SFRs to Security Objectives....41
Table 6: SFR Sufficiency Analysis....43
Table 7: SFR Dependencies....45
Table 8: Security Assurance Requirements....46
Table 9: SFR to TSS References....51

Illustration Index

Figure 1: Schematic architecture of the hardware platform, TOE and user partitions....11
Figure 2: Configuration examples with proxies....18
Figure 3: Proxy policies....19

1 Introduction

The purpose of this Security Target (ST) is to define the evaluated and certified properties of the L4Re Secure Separation Kernel CC 1.0.2. Throughout this document, the L4Re Secure Separation Kernel CC 1.0.2 will be abbreviated as L4Re SSK. Further, we want to give potential users the confidence that the properties of L4Re SSK can be used along with other application software to build an integrated TOE (Target of Evaluation), consisting of the L4Re SSK and the user's own software components.

1.1 Security Target Identification

Title: Security Target for L4Re Secure Separation Kernel CC 1.0.2

Version: 2.1700

Status: ready for publication

Publication Date: 2026-01-26

Authors: Kernkonzept GmbH

BSI Administration ID: BSI-DSZ-CC-1177-v2-2026

CC-Version: 3.1 Revision 5

Keywords: Microkernel, Operating System, Separation Kernel, Hypervisor, Open Source, L4Re Operating System Framework

1.2 TOE Identification

The TOE is the L4Re Secure Separation Kernel CC 1.0.2, abbreviated as L4Re SSK throughout this document.

1.3 TOE Overview

The TOE is the L4Re Secure Separation Kernel CC 1.0.2 (L4Re SSK). L4Re SSK is a distribution of the open-source L4Re Operating System Framework. As such it is based on the L4Re Microkernel. The L4Re Microkernel is a 3rd-generation microkernel with a state-of-the-art capability-based mandatory access-control security model. It allows the separation of applications into different security domains, information flow control, and access-controlled dynamic assignment of resources and communication channels. Further the L4Re Microkernel supports static workloads alongside dynamic workloads, which allows to start, restart and shutdown applications during runtime, as well as re-assigning resources.

L4Re SSK is configured to act as a separation kernel, to provide the security features claimed in this ST. The TOE supports native applications as well as virtual machines (VMs). Access to every resource including but not limited to memory, hardware devices and CPU cores is protected by capabilities. Applications and VMs can only access a resource if they possess a capability with suitable permissions for that resource.

A capability is an unforgeable token of authority which is managed and protected by the L4Re Microkernel. It consists of a reference to a resource and access permissions. An application may invoke a function of a certain resource by sending a message referencing that resource's capability

as the destination of that message. With capabilities L4Re SSK allows the efficient implementation of the principle-of-least-authority (POLA).

On the basis of the capability-based access control, the TOE allows to control the information flow between single applications or groups of applications, called *compartments* hereafter. The term *compartment* refers to one initial application together with all applications that are started directly and indirectly by this initial application. Initial applications are defined by L4Re SSK's start-up configuration.

Within the L4Re SSK an application is a possibly multi-threaded program, where all threads of one application share the same access rights. Applications can be developed with a separate SDK which is not part of the TOE.

1.3.1 TOE Type

The TOE type is a separation-kernel operating system.

1.3.2 Intended Method of Use

L4Re SSK requires a start-up configuration provided by means of a script (written in Lua) to the application loader Ned. The script starts the initial applications and equips them with capabilities. The initial capability distribution defines the permissions of the initial applications, allocates devices to them and sets up the initially shared resources, especially the initial capability and data channels. Depending on permissions, each initial application may start additional applications. L4Re SSK is able to enforce separation properties between all applications, regardless of how they are started. However, this ST only claims separation properties between compartments.

The evaluated configuration for L4Re SSK sets forth rules how to create resources and devices as well as how to assign them to compartments. Other Lua configurations are not allowed and may lead to a system that violates one or more of the security features outlined in the following sections.

To realize a product based on L4Re SSK an integrator must choose a platform that meets the TOE requirements, see Section 1.3.5, and a set of initial applications, each representing one compartment, to be started on L4Re SSK. The integrator must decide which compartments shall have access to which platform devices and about the communication channels between the compartments. For this purpose the integrator may use any number (including zero) of the optional TOE components. The intended compartment configuration must be checked for consistency and realizability with respect to the rules described in *L4Re Configuration Guidance.* Additionally, it must be checked if the required separation properties between the compartments are enforced. To achieve the desired communication channels and separation properties it may be necessary to design and implement product specific communication proxies. The integrator must decide about how to protect the integrity of the L4Re SSK installation on the platform and how to roll out updates in response to L4Re SSK security advisories with an appropriate rollback prevention. This may involve an appropriate key management process to be implemented by the integrator. Finally the integrator must create a L4Re SSK configuration adhering to the *L4Re Configuration Guidance* and install the TOE based software stack on the platform, see also Section 1.4.2.3.

1.3.3 Major Security Features

1.3.3.1 Information Flow Control

L4Re SSK implements mandatory, capability-based access control. Each object controlled or implemented by L4Re SSK is represented by a capability. The communication channels available between applications can effectively be controlled by the initial capability distribution. Additional details are provided in Section 1.4.2.1.

1.3.3.2 Separation of Applications and Compartments

L4Re SSK ensures the separation of applications by enforcing capability-based, mandatory access control for all applications. An application can only access a resource if it possesses a capability.

Additional details are provided in Section 1.4.2.2.

1.3.3.3 System Management

L4Re SSK is an operating system that is intended to ensure the separation of applications while permitting flexible workloads and the reconfiguration of the system during run time. Further details are provided in Section 1.4.2.3.

1.3.4 Product Types

Using the security features outlined in Section 1.4.2 together with trusted applications (which are not part of the TOE), a variety of products can be realized on the basis of the TOE, such as data diodes, secure exchange gateways, secure VPN gateways or secure endpoint devices. This is not a complete list of products and other solutions such as secure cloud workloads and safety applications can be realized with the TOE.

1.3.5 Required Hardware and Software

The TOE requires hardware systems with the following components – additional hardware is considered a form factor which does not affect the security functionality of the TOE:

- x86_64 Intel CPU with any microarchitecture starting from Broadwell up to and including Meteor Lake and, additionally, with support for VT-x with EPT, and Intel IOMMU. If available, the latest microcode update must be applied by the BIOS.
- ARMv8 CPU: NXP LX2160A with support for ARM virtualization

The following assumptions about the execution environment are made:

- The TOE installation is protected against unauthorized modifications.
- If the exploitation of covert channels via an L1/L2 cache or TLB shall be prevented, L4Re SSK must be configured to execute the compartments to be separated on different CPU cores that do not share a common L1/L2 cache and TLB (for more details see the guidance listed in Section 1.4.4.2).
- ARM virtualization and Intel virtualization (VT-x) are allowed, but there will be made no claims about security requirements in this Security Target. There are three variants of L4Re SSK available, one per supported platform and configuration:
  - LX2160A with support for the SMMU

- Intel 64 with support for VT-x and the IOMMU
- Intel 64 with support for the IOMMU but without support for VT-x

The binaries are included in the corresponding archives of the TOE (see Section 1.4.4.2).

1.4 TOE Description

This section provides a general description of the TOE, including physical boundaries, security functions, and relevant TOE documentation and references.

1.4.1 System Architecture

The L4Re Secure Separation Kernel CC 1.0.2 is configured to work as a separation kernel. The L4Re SSK is a microkernel-based operating system where the functionality is split among the operating system kernel and multiple user land applications. This design (called the microkernel principle) provides benefits over traditional monolithic designs. Only the L4Re Microkernel runs in the most privileged CPU mode where it provides the basic services needed for isolating user-land applications from each other as well as to implement use-case specific services and policies in those applications. This system architecture combined with the state-of-the-art capability-based mandatory access control allows L4Re SSK to efficiently implement the Principle of Least Authority (POLA) which enables Zero Trust. The L4Re Microkernel guarantees address-space separation (see Section 1.4.2.2) between all user land TOE components as well as temporal isolation.

L4Re SSK implements the middle ware which provides the mechanisms and services that an OEM needs to build its own product or solution for an end user. For that the OEM must instantiate so called user partitions or compartments according to the configuration guidance provided with L4Re SSK. Please see Figure 1 for a schematic architecture of the hardware platform, the TOE and example user partitions.

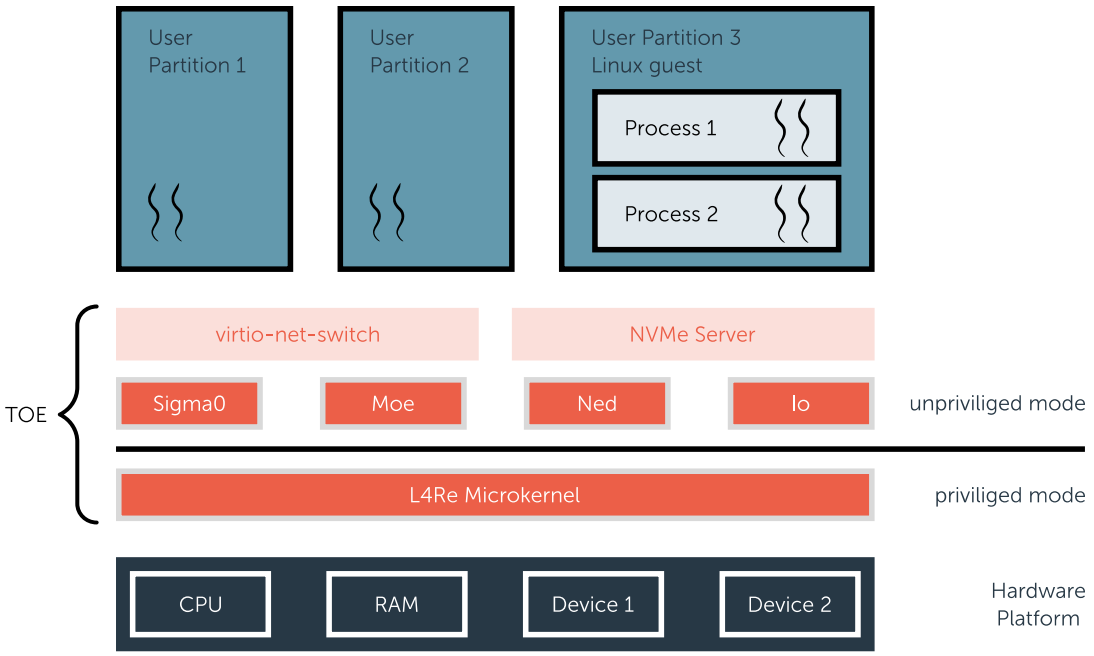


Figure 1: Schematic architecture of the hardware platform, TOE and user partitions. This is a schematic architecture. Between the hardware platform and the user partitions (light blue) runs the TOE. The components in light orange are optional and may be used depending on an OEM's use case. Each box denotes address space separation between the individual components. The processes inside user partition 3 are not separated by the L4Re Microkernel in the sense of this ST but they run in different address spaces as far as the guest operating system ensures this.

A user partition for example can be a virtual machine that hosts a virtualized Linux guest operating system or native applications. The L4Re Operating System Framework offers a wide variety of applications apart from components belonging to the TOE. Additionally, the L4Re Operating System Framework provides a software development kit for developing customized L4Re applications, for instance customized cryptographic components. However, the security features claimed in this Security Target are solely provided by the TOE and the underlying hardware platform.

The TOE comprises of:

L4Re Microkernel	The L4Re Microkernel provides Tasks, Threads, and a capability-based access control mechanism to implement the separation types described in Section 1.4.2.2. It provides basic mechanisms like CPU scheduling, IPC, and resource delegation. The L4Re Microkernel always runs in a higher-privileged CPU mode than the user-level components of the TOE.
Sigma0	Sigma0 is also called the root pager and is a special application running on the L4Re Microkernel. It is responsible for the resolution of page faults of Moe (the root task, see below). It provides capabilities to memory (usually RAM), MMIO, and I/O ports (x86 only) on a first-come first-served basis. This component is a user-level component and runs in a less-privileged CPU

	mode.
Moe	The root task. Moe is responsible for starting the initial application (Ned) and it provides essential services for all non-privileged components (except for Sigma0). Moe eagerly acquires all memory (RAM) from Sigma0 and makes it available to other applications. This component is a user-level component and runs in a less-privileged CPU mode.
Ned	Ned is responsible for starting all initial applications including Io. It sets up the initial communication channels according to a configuration provided as Lua scripts in boot modules. This component is a user-level component and runs in a less-privileged CPU mode.
Io	Io is the device and platform manager. It provides secure access to platform devices and I/O resources. Lua scripts that are provided as boot modules define virtual device trees (called virtual bus = vbus) to which Io maps physical platform devices. The mapping may include the filtering or exclusion of device features in order to ensure proper separation. Each virtual bus is represented by a capability that only provides access to the devices of the corresponding device tree. This restricts the access to devices according to the capability distribution set up by Ned. This component is a user-level component and runs in a less-privileged CPU mode.

Two optional L4Re applications are part of the TOE. They are so called communication proxies (see Section 1.4.3). If the use case doesn't require the functionality of these two applications they can be removed from the system configuration.

virtio-net-switch	The virtual network switch connects multiple clients with a virtual network connection. It uses virtio as the transport mechanism. Each virtual switch port implements the host-side of a virtio network device. The virtual network switch can be setup to feature exactly one monitor port. All traffic passing through the switch is mirrored to the monitor port but a client connected via the monitor port cannot send data through this port. An optional packet filter can be configured and implemented to filter data sent to the monitor port.
NVMe server	The NVMe server is a driver for PCI Express NVMe controllers. It is capable of exposing entire disks (i.e. NVMe namespaces) (by serial number and namespace identifier) or individual partitions (by their partition UUID) of a storage device to clients via the Virtio block interface. The server consists of two parts. The first one is the hardware driver itself that takes care of the communication with the underlying hardware and interrupt handling. The second part implements a virtual block device and is responsible to communicate with clients. The virtual block device translates commands it receives into NVMe requests and issues them to the hardware driver.

Apart from the components belonging to the TOE, the L4Re Operating System Framework offers a variety of other components that can be used inside compartments, for instance virtual machines as provided by Uvmm, a virtual machine monitor supporting fully virtualized guests. Additionally, the L4Re Operating System Framework provides a software development kit for developing customized L4Re applications, for instance customized cryptographic components. The security features claimed in this Security Target are solely provided by the TOE and the underlying hardware platform.

1.4.2 Major Security Features

L4Re SSK implements capability-based, mandatory access control. An application can only access a resource if it possesses a capability referencing that resource. Here, resources include all architecturally visible hardware resources (e.g. memory pages, hardware interrupts), all system services provided by the L4Re SSK (e.g. threads, data objects, communication channels, memory allocators) and all services implemented by applications running on L4Re SSK. An acting entity that possesses a given capability can perform operations on that capability by invoking the capability. The allowed operations to be exercised on a capability depend on the type of the capability, i. e., which object the capability references, and the capability's access permissions.

The security domains in L4Re SSK are the applications. For each application, L4Re SSK manages a private capability space and a collection of threads that execute in that application. A thread is an executable piece of code and is, therefore, considered the subject, which is also called the acting entity. The L4Re SSK object to manipulate an application is called task. Tasks and threads are also objects that can be controlled using their respective capabilities.

L4Re SSK features kernel resource management and IOMMU support to protect the TOE and all applications from malicious or malfunctioning DMA-capable devices or device drivers. Kernel resource management limits the amount of kernel objects that an application can create and, thereby, can prevent a single application from monopolizing all kernel memory.

Details of the major security features are specified in the following sections.

1.4.2.1 Information Flow Control

L4Re SSK supports three different kinds of communication channels between applications:

Capability channel	Capability channels permit the transfer of capabilities. The typical example is an IPC-gate that allows to transfer capabilities and data.
Bidirectional data channel	Shared memory and semaphores allow to transfer data and signals in both directions but do not permit the transfer of capabilities. As a special case, shared access to a physical device is considered as a bidirectional data channel in this document.
Unidirectional data channel	Semaphores and IRQs with restricted permissions do only permit to trigger signals but not to receive signals or to transfer capabilities.

An application that possesses a capability for a capability channel may attempt to send any of its own capabilities over that channel. The application at the other end of the channel will receive copies of the transmitted capabilities if it had agreed to receive capabilities before. The permissions at the receiver are those of the sender, possibly reduced by a permission mask specified by the

sender and the capability channel. A capability channel thus permits to set up additional communication channels.

Inter process communication (IPC) by invoking communication objects permits capability transfer by default. However, each IPC partner can prevent the capability transfer. This is obvious for the sender but also possible for the receiver by suitably configuring receive windows. Using this mechanism, TOE components protect themselves against unwanted capability transfers from other L4Re applications. L4Re permits to effectively restrict and control capability transfers and communication between tasks. On one hand, sender and receiver can prevent the capability transfer during IPC as stated above.

On the other hand, capability transfer between applications is only possible via thread, task, and IPC-Gate capabilities. Therefore, by an appropriate distribution of these capabilities, capability transfer can be allowed/disallowed without relying on the tasks outside the TOE. Nevertheless, it is still possible to communicate without capability transfer via shared memory, semaphores, IRQs, devices, or I/O ports. L4Re SSK allows the creation of such data channels between tasks which are solely based on shared memory, semaphores, IRQs, devices, or I/O ports. All these data channels require some shared capability. For instance, for shared memory the involved applications must at least both possess a memory capability referencing the same memory frame. Therefore, via controlling capability channels, L4Re SSK also permits the control of communication channels.

Capabilities for memory pages and —on x86— I/O ports as well as the capabilities for certain special objects are created during L4Re SSK startup. For all other objects, a new capability referencing the object is generated when the object itself is created and given to the creator of the object. For example, when a new task is created, the capability referencing that task with full access permissions is created. For a second example, when additional main memory is allocated, a capability referencing the new dataspace with full access permissions is created.[1] After creation of a new object, only the application that created the object possesses a capability referencing the new object. That application can of course share the new capability with other applications, including L4Re SSK services, via existing capability channels. When an object is destroyed, all capabilities referencing that object are destroyed as well.

An application can only create objects if it possesses a capability referencing some object that offers object creation as function. Such an object is called *Factory* in L4Re SSK. L4Re SSK provides two initial factories that are required by each application. The first one is the kernel factory which is implemented by the L4Re Microkernel. It allows to create kernel objects such as tasks, threads, IPC gates or semaphores. The second one is the user factory which is implemented by Moe in L4Re SSK.

1.4.2.2 Separation of Applications and Compartments

The TOE provides three different kinds of separation between applications or compartments.

Address-space separation	Address-space separation is the default minimal separation of applications. Except for explicitly configured shared memory, each application has access to its private memory only. Each application also has its own (private) capability space, whose integrity is protected by the TOE. Limited by access control, applications can share their own capabilities with other applications. Each application can

[1] A Dataspace is a user-level object that encapsulates a chunk of memory.

	effectively protect itself against receiving capabilities or overwriting slots in its capability space when being granted access rights.
Capability separation	Applications can be configured such that they can freely exchange any data but cannot exchange capabilities. The data can be exchanged via shared memory or via special communication channels (e. g., semaphores) that cannot transfer capabilities. Capability separation is useful if exchange of data between two compartments is permitted but the integrity of the compartments should be protected by the TOE, for instance, because one does not trust the involved compartments to correctly use the TOE features to protect their integrity themselves.
Strong separation	Applications and compartments can also be configured such that they can exchange neither data nor capabilities using the features of the TOE. Strong separation restricts only the direct transfer of data or information between two compartments, that is, if two strongly separated compartments are both connected to a third compartment, then they may exchange data via the third compartment (see section 1.4.3)

The TOE does not enforce the absence of side channels or hidden channels for compartments that enjoy any of the previous separation properties and this ST does not claim that such channels do not exist between separated compartments. However, the TOE provides means to restrict the execution of any compartment to a specified subset of CPU cores to limit side and covert channels in hardware. An integrator may, for instance, decide to let two compartments execute in a time-shared manner on the same CPU core, if both compartments belong to the same security domain. However, the integrator may also decide to pin two compartments to separate cores to eliminate side and covert channels via shared L1/L2 caches, TLBs and other CPU resources.

The only system call in L4Re SSK is object invocation, which calls objects in the L4Re Microkernel with an object specific payload. IPC is built by invoking a communication object in the L4Re Microkernel. Using this communication object, messages, capabilities and access rights can be transferred to other threads. A thread may use each capability of its application as target of an object invocation. A thread may also wait for incoming IPC messages from other threads that have access to a capability referencing the receiving thread or a resource it implements. The sending thread may add arbitrary capabilities of its application to the IPC message[2]. If the IPC-receiving thread sets up its state to receive capabilities, the capabilities in the message are transferred (mapped) to the receiver. Depending on the invoked function, the IPC message may need to contain certain capabilities as required arguments. In order to prevent unintended exchange of capabilities and thereby unintended sharing of resources between two applications, the system integrator must chose an initial configuration in which the two applications are not connected via a capability channel and where they cannot exchange capabilities via a third, untrusted application.

The following resources are fully controlled by L4Re SSK:

Memory resources	During startup L4Re SSK creates a memory capability for each page of main memory. L4Re SSK configures the hardware memory-management unit (MMU) such that each application can only access those memory frames for

[2] IPC message storage size limits apply.

	which it possesses appropriate memory capabilities. To grant access to memory, memory capabilities can be transferred via capability channels to different applications. To manage regions of memory more conveniently, L4Re SSK provides the abstraction of dataspaces. The free memory (RAM) is managed by Moe. Every application with an appropriate Moe capability can create a new dataspace from the free memory available to this application that is guaranteed to be disjoint from all existing dataspaces.
Cores	L4Re SSK controls all cores available in the system. Acting entities (threads) can be assigned to cores. It is permissible to define an n:m relationship where zero or more acting entities are assigned to one core. Yet, only one thread is executed on one core at any given time. During a context switch, the CPU resources, such as general purpose registers, are saved for the current thread and re-loaded for the newly scheduled thread. Conversely, one acting entity is allowed to be scheduled on one or more cores but only on one core at a time. L4Re SSK does not automatically migrate acting entities between cores. However, migration can be implemented in an application outside of the TOE using the APIs provided by L4Re SSK. As migration is explicit, covert channels can be controlled by pinning acting entities to specific cores.
Physical devices	Access to all physical devices of the system is managed by L4Re SSK via vbus capabilities. Each vbus capability provides access to all devices contained in its associated set of devices. A vbus capability permits to enumerate the devices available on this virtual bus, to get access to the device's resources (i.e. interrupts, MMIO regions, I/O ports) and to configure DMA memory for a particular device. Applications without a vbus capability cannot access any physical devices. By using appropriately configured device sets in conjunction with capability separation and strong separation, an integrator can achieve that certain devices are inaccessible, that certain applications have exclusive access to a certain device, or that devices are shared between certain applications.

The scheduling of threads in the TOE is strictly priority-based. This allows the execution of multiple threads on one CPU if not excluded by the configuration.

1.4.2.3 Configuration and System Management

The system integrator defines the behavior of a system using the L4Re SSK. The system integrator is a user who has access to a set of configuration files while L4Re SSK is non-operational. Since L4Re SSK is non-operational whenever the system integrator accesses the configuration files, L4Re SSK does not control the role of the system integrator.

L4Re SSK ensures the separation of applications while permitting flexible workloads and the reconfiguration of the system during run time. All the functionality necessary for reconfiguration is governed by capabilities, in particular the creation and the start of new applications, the creation of capability and communication channels and connecting existing applications via new channels. By choosing an appropriate initial capability distribution, the integrator can limit this functionality to a particular subset of the applications or disable it completely.

In L4Re SSK the Lua scripting language is used to express the initial configuration. This gives the system integrator the power and flexibility of a programming language to express the required system configuration. The permitted configurations are limited by the rule set in the *L4Re Configuration Guidance*. This configuration guidance also contains the necessary documentation on how to write these Lua scripts.

The configuration is split into at least two different scripts that are processed by the Ned and Io components during boot time. The first script can be compared to an init script in e.g. a Linux system. It describes how the initial applications should be started and how the initial applications can communicate with each other. This script is processed by Ned. The second script is processed by Io. It defines all vbus'es and associates a device set with each vbus thereby expressing the allocation of hardware devices to those compartments that have access to a vbus capability. On platforms where the system firmware doesn't provide the system's devices via the Advanced Power and Configuration Interface (ACPI) the hardware device tree also needs to be expressed using a Lua script and passed on to Io.

In order to configure a system based on L4Re SSK an integrator needs to decide which compartments should run on top of L4Re SSK, which of those should be able to exchange capabilities or data in an arbitrary run of the system and which compartments should have access to the available hardware devices.

Following the rules in the configuration guidance, the integrator can configure an initial configuration of compartments and communication channels. L4Re SSK will then enforce the separation properties (described in Section 1.4.2.2) in any run of the system. The compartments running on L4Re SSK do not have access to the Lua scripts.

Certain configurations cannot be realized with L4Re SSK. For instance, if the integrator does not trust any of the applications A, B and C, then a configuration in which A and B as well as B and C can exchange data but A and C must not be able to exchange data is impossible, because the untrusted B can always forward data. The integrator may solve such problems by using communication proxies with certain properties (see Section 1.4.3 below) and ensuring that all communication between A and B as well as B and C is routed through such a proxy.

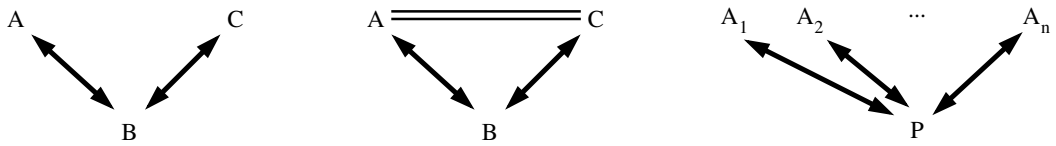
As a final step of configuring L4Re SSK, the system integrator needs to assemble a boot image consisting of L4Re SSK, the compiled binaries of the applications that shall run on L4Re SK and the configuration scripts. The boot image must be installed on the target system such that its integrity is protected against unauthorized modifications.

1.4.2.4 Secure Boot

L4Re SSK supports secure boot on x86 via GRUB2 and either UEFI or coreboot, see the *L4Re Secure Boot Guidance*[3]. However, this feature is not claimed in this ST. The integrator is responsible for ensuring that any installation of the TOE is appropriately protected against unauthorized modifications.

1.4.3 Communication Proxies

[3] Available on request.



a) initial configuration b) final configuration c) generic proxy configuration

Figure 2: Configuration examples with proxies.
Bold double arrows depict capability channels, double lines depict shared memory. Part a) shows an initial configuration in which the compartments A and C are strongly separated. Part b) shows a later configuration, in which B has established shared memory between A and C. Part c) shows a generic proxy P serving all A_i.

Using the fine-grained control over the capability distribution and the different communication channels combined with customized L4Re SSK applications, a system integrator can realize a wide variety of products based on the TOE. Consider an initial configuration with three compartments *A*, *B* and *C*, where *A* and *B* as well as *B* and *C* are connected with a capability channel, see Figure 2a. In this initial configuration *A* and *C* are strongly separated in the sense of Section 1.4.2.2. However, the separation of *A* and *C* for the remainder of the run time depends on the behaviour of *B*. *B* could, for example, exchange memory capabilities with *A* and *C* to establish shared memory between *A* and *C*, downgrading their separation to capability separation, see Figure 2b. In such a situation, where two or more compartments A_1, A_2, …, A_n only have access to a communication channel to a dedicated, extra compartment *P* and where the long term separation of the A_i depends on *P*, *P* is called a *communication proxy*, see Figure 2c.

A communication proxy *P* connecting *n* compartments A_i as in the previous paragraph can implement the following policies.

- **strong separation without data exchange:** The proxy *P* does neither establish any communication channels between the A_i nor does it forward any data from one compartment to another one. Note that a proxy *P* implementing strong separation may exchange certain capabilities with each of the connected compartments A_i, for instance to establish separate regions of shared memory for fast communication with some or all of the A_i. See Figure 3a for an example with two compartents A and B.
- **strong separation with data exchange according to an explicitly configured policy:** As before, the proxy *P* does not establish any communication channels between the A_i. However, it forwards data between the connected compartments A_i according to a policy implemented in *P*. For instance, *P* may forward data only after encryption or decryption or it may forward data only in a certain direction. See Figure 3b.
- **capability separation:** The proxy *P* establishes bidirectional data channels between all the A_i but does not establish any capability channel between any of the A_i. See Figure 3c.
- **no separation:** The proxy may establish arbitrary communication channels between the connected compartments or forward data in arbitrary ways. See Figure 3d.

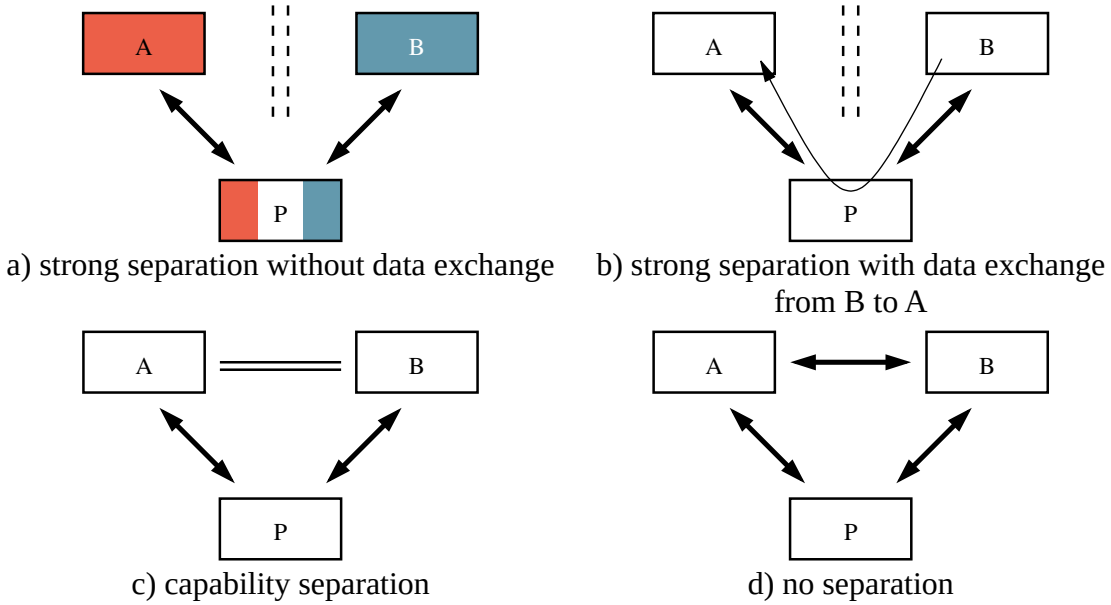


Figure 3: Proxy policies.

The pictures show the final configuration of a proxy P serving the compartments A and B, which started in an initial configuration as shown in Figure 2a. Bold double arrows depict capability channels, double lines depict shared memory, double dashed lines depict separation and a single arrow depicts data exchange. Part a shows strong separation without data exchange. Here the proxy P strictly separates its clients internally, such that no communication between A and B is possible. Part b shows strong separation with data exchange in one direction. Here the proxy P forwards data from B to A but not vice versa. Part c shows capability separation, where the proxy establishes shared memory between A and B, such that A and B can communicate without control of the proxy. Part d shows no separation, where the proxy establishes a capability channel between A and B, such that A and B can exchange capabilities without control of the proxy.

From the point of information flow control, a proxy implementing strong separation with free data exchange between all connected compartments, a proxy implementing capability separation, and a configuration without a proxy but with a bidirectional data channel between all compartments are all equivalent. Depending on the connected compartments, one solution might however be easier to implement.

L4Re SSK contains two communication proxies that can optionally be used.

- **NVMe server:** The NVMe server implements strong separation without data exchange. It can be used to provide persistent storage to several strongly separated compartments. Each compartment is assigned a separate partition, that no other compartment has access to.
- **virtio-net-switch:** The virtio-net-switch implements a virtual network switch based on virtio. It implements strong separation with free data exchange between all connected compartments.

In addition to these two communication proxies, L4Re SSK provides the *Development Guidance for a Compartment Communication Proxy based on L4Re*, that specifies sufficient conditions for a proxy to implement one of the previously enumerated policies. The guidance plus the separate

L4Re Operating System Framework SDK can be used to implement and certify additional communication proxies.

1.4.4 TOE Boundaries

1.4.4.1 Logical Boundary

The major security features of the TOE are described in Section 1.4.2.

1.4.4.2 Physical Boundary

The TOE is the L4Re SSK. In Figure 1, each orange box is within the TOE physical boundary. All parts with different colors belong to the TOE environment (CPUs, RAM, firmware such as a BIOS, devices, L4Re applications, etc.). Consequently, no hardware belongs to the TOE.

The TOE also includes the TOE User Manuals. A list of the TOE User Manuals is given below.

There are three variants of the TOE. One for the supported ARM board and two for x86, one with and one without support for virtualization (VT-x). Hence, the TOE is provided through the following compressed zip-archives:

Type	Name	Version	Form of delivery
Software	l4re_ssk_cc_lx2160a.zip	1.0.2	Compressed zip archive for LX2160A with SMMU
Software	l4re_ssk_cc_x86_virt.zip	1.0.2	Compressed zip archive for x86 with VT-x enabled
Software	l4re_ssk_cc_x86_novirt.zip	1.0.2	Compressed zip archive for x86 with VT-x disabled
Documentation	l4re_ssk_cc_customer_doc.zip	1.0.2	Compressed zip archive of TOE documentation
Documentation	l4re_ssk_cc_interface_and_usage_documentation.zip	1.0.2	Compressed zip archive containing the L4Re Interface and Usage Documentation; separately available

These zip-archives can be identified by the following Git-object hashes.

Name	Git-object hash
l4re_ssk_cc_lx2160a.zip	6bceb5f05015819cb9d70e2ce9391d17dfc83dde
l4re_ssk_cc_x86_virt.zip	2de780b0d6b09db6b24008cc54a30265ee50e191
l4re_ssk_cc_x86_novirt.zip	05d20a993ba6166f228884b33c2715cc42abf57d
l4re_ssk_cc_customer_doc.zip	be2390ec82decb6d7be4df4adcf2cdfd94150459

l4re_ssk_cc_interface_and_usage_documentation.zip	3ffa6276da70dce7a1287ee32b1812c18599e7f8

The TOE User Manuals in the documentation archive are provided as PDF files through:

Type	Name	Version	Short description
Document	L4Re Configuration Guidance.pdf	15	L4Re Configuration Guidance
Document	Development Guidance for a Compartment Communication Proxy based on L4Re.pdf	8	Development Guidance for a Compartment Communication Proxy based on L4Re

The TOE is delivered to the customer either via remote access to the respective directory or as download, in a way preserving authenticity, confidentiality and integrity.

1.4.4.3 Configurations

The TOE configuration enforcing the System Security Policy (SSP) is defined via a set of configuration files written in Lua. The configuration data uniquely defines the SSP consisting of the configuration choices made by the integrator regarding resources and devices and their allocation to compartments. The SSP defines compartments, sets their resources and devices, and defines the allowed communication channels.

1.4.4.4 Security Policy Model

The security policy for the TOE is defined by the security functional requirements in Chapter 6. The following is a list of the subjects and objects referenced by the policy.

Subjects:

- A thread executing as part of an application, more general an application or compartment.

Named objects: See objects in Table 4.

TSF data consists of:

- Configuration data: data determined in accordance with the configuration guidance and used by the TSF to enforce the SSP.
- Run-time data such as security attributes.

User data consists of:

- All data maintained by compartments with the named objects assigned to the compartment.
- Executables to be executed in a compartment.

2 Conformance Claims

2.1 Conformance with CC Parts 2 and 3

This Security Target is CC Part 2 conformant and CC Part 3 conformant, with a claimed Evaluation Assurance Level of EAL4, augmented by ALC_FLR.3.

The Common Criteria [CC] version 3.1 revision 5 are the basis for this conformance claim.

2.2 Conformance with Protection Profiles

This Security Target does not claim conformance to any Protection Profile.

3 Security Problem Definition

3.1 Threats

Threats to be countered by the TOE are characterized by the combination of an asset being subject to a threat, a threat agent and an adverse action.

3.1.1 Assets

Assets to be protected are:

Asset Name	Description	Security Properties to be Preserved
Memory	RAM or ROM memory	Confidentiality, integrity, availability
CPU cores	Set of CPU cores allocated to each compartment and configured by the integrator	Integrity
TSF data	Configuration data: data used by the TSF to enforce the System Security Policy (SSP, Section 1.4.4.3) Run-time data such as security attributes.	Confidentiality, integrity
Device	A device with memory mapped I/O or I/O ports may be a real device or a virtualized device. A real or virtualized device may only be shared between compartments when this is deliberately configured. Remapping of a device to a different compartment can only be allowed if the device looses its entire state during a power-cycle or the two compartments are not strongly separated .	Confidentiality, integrity, availability

Table 1: Assets

3.1.2 Threat Agents

Threat agents are subjects within an untrusted compartment.

3.1.3 Threats countered by the TOE

T.DISCLOSURE A threat agent reads an asset of which the property confidentiality shall be maintained according to Table 1.

T.MODIFICATION A threat agent writes an asset for which the property integrity shall be maintained according to Table 1.

T.DEPLETION	By consuming resources of which the property availability shall be maintained according to Table 1 a threat agent makes these resources unavailable to the TOE itself and/or to tasks/compartments using/owning them.

3.2 Organizational Security Policies

Organizational security policies are not defined.

3.3 Assumptions

The specific conditions below are assumed to exist in the TOE environment.

A.ENVIRONMENT	The underlying hardware, firmware and bootloader needed by the TOE to guarantee secure operations fulfil the requirements, as explained in the TOE User Manuals and stated in Section 1.3.5. They are working correctly and have no undocumented or unintended security critical side effect on the functions of the TOE.
A.PHYSICAL	The IT environment provides the TOE with appropriate physical security that is commensurate with the value of the IT assets protected by the TOE.
A.NOEVIL	The integrators are trustworthy, act according to the guidance documentation and are sufficiently qualified for this task.

4 Security Objectives

The following sections describe the security objectives, which are concise, abstract statements of the intended solution to the problem posed in the security problem definition (see Chapter 3). The set of security objectives for a TOE form a high-level solution to this security problem. It is divided into two part-wise solutions: the security objectives for the TOE, and the security objectives for the TOE's operational environment.

4.1 Security Objectives for the TOE

The following objectives are defined for the TOE.

O.CONFIDENTIALITY	For each asset that requires confidentiality protection according to Table 1, the TOE shall preserve its confidentiality.
O.INTEGRITY	For each asset that requires integrity protection according to Table 1, the TOE shall preserve its integrity.
O.AVAILABILITY	For resources assigned to tasks / compartments and to TSF data that requires availability protection according to Table 1, the TOE shall preserve their availability.

4.2 Security Objectives for the Operational Environment

The following objectives are to be met by the operational environment of the TOE.

OE.HARDWARE	The underlying hardware, firmware and bootloader needed by the TOE to guarantee secure operations fulfil the requirements, as explained in the TOE User Manuals and stated in Section 1.3.5. They are working correctly and have no undocumented or unintended security critical side effect on the functions of the TOE.
OE.PHYSICAL	The IT environment provides the TOE with appropriate physical security that is commensurate with the value of the IT assets protected by the TOE.
OE.NOEVIL	The integrators are trustworthy, act according to the guidance documentation, and are sufficiently qualified for this task.

4.3 Rationale for Security Objectives

The following tables provide a mapping of security objectives to the threats and assumptions, illustrating that each security objective covers at least one threat or assumption and that each of those is covered by at least one security objective.

Threat / OSP	**Rationale**
T.DISCLOSURE	T.DISCLOSURE is countered by O.CONFIDENTIALITY as the TOE will protect the assets against unauthorized read accesses.
T.MODIFICATION	T.MODIFICATION is countered by O.INTEGRITY as the TOE will

Threat / OSP	Rationale
	protect the assets against unauthorized modification accesses.
T.DEPLETION	T.DEPLETION is countered by O.AVAILABILITY as the TOE will actively keep the resources operation alive and available for tasks / compartments using/owning them.

Table 2: Coverage of Security Objectives for the TOE

Threat / Assumption	Rationale
A.ENVIRONMENT	A.ENVIRONMENT is directly upheld by OE.HARDWARE.
A.PHYSICAL	The assumption on the IT environment to provide the TOE with appropriate physical security that is commensurate with the value of the IT assets protected by the TOE is covered by: • OE.PHYSICAL requiring that the IT environment provides the TOE with appropriate physical security, commensurate with the value of the IT assets protected by the TOE.
A.NOEVIL	The assumption on the IT environment that the integrators are trustworthy, act according to the guidance documentation and are sufficiently qualified for this task is covered by: • OE.NOEVIL requiring that the personnel configuring the TOE and those installing and operating the TOE are trustworthy, act according to the TOE guidance, and are sufficiently qualified for this task.

Table 3: Coverage of Security Objectives for the TOE environment

5 Extended Components Definition

There are no extended components in this ST.

6 Security Requirements

This section defines security functional requirements (SFRs) and security assurance requirements (SARs), which apply for the TOE.

The following styles of marking operations are applied:

- Assignments and selections are marked in **bold face font**.
- Iterations are marked by appending a suffix to the SFR identification.
- Refinements indicating additions are marked in ***bold and italic face font***.
- Refinements indicating removals are marked as ~~crossed out~~.

6.1 Security Functional Requirements

6.1.1 User Data Protection (FDP)

6.1.1.1 FDP_ACC.2/ME Complete Access Control – Memory

FDP_ACC.2.1/ME The TSF shall enforce the **memory access control policy** on

- **subjects: compartments,**
- **objects: memory**

and all operations among subjects and objects covered by the SFP.

FDP_ACC.2.2/ME The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP.

6.1.1.2 FDP_ACC.2/CC Complete Access Control – CPU Core

FDP_ACC.2.1/CC The TSF shall enforce the **CPU access control policy** on

- **subjects: compartments,**
- **objects: CPU cores**

and all operations among subjects and objects covered by the SFP.

FDP_ACC.2.2/CC The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP.

Application Note: This access control policy concerns objects where the access control is not managed via capabilities.

6.1.1.3 FDP_ACC.2/DE Complete Access Control – Device

FDP_ACC.2.1/DE The TSF shall enforce the **device access control policy** on

- **subjects: compartments,**
- **objects: PCI devices**

and all operations among subjects and objects covered by the SFP.

FDP_ACC.2.2/DE The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP.

Application Note: On the LX2160A platform, PCI devices are not supported by L4Re SSK.

6.1.1.4 FDP_ACC.2/OB Complete Access Control – Objects

FDP_ACC.2.1/OB The TSF shall enforce the **objects access control policy** on

- **subjects: compartments,**
- **objects: objects of Table 4**

and all operations among subjects and objects covered by the SFP.

FDP_ACC.2.2/OB The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP.

Object	Description	Relevance for the assets
Memory pages	Memory pages can be obtained via dataspaces and can only be accessed if appropriate memory capabilities are present in the respective task. Memory pages are part of the ROM or RAM memory defined by dataspace objects.	Memory
Tasks	A tuple consisting of a set of threads and a capability space. The threads are those threads that run “inside” this task. The capability space contains all capabilities that are accessible to the threads of this task, including capabilities for memory, I/O ports, memory mapped devices, and capabilities referring to software objects, such as, dataspaces.	TSF data, Devices, Memory,
Threads	Management object for an executable piece of code including the necessary control information, such as instruction pointer, stack pointer, and other CPU register content. Threads are the only subjects or acting entities. The Thread object can manage native L4Re threads as well as virtual CPUs in fully or paravirtualized guests.	CPU cores
Moe-Scheduler	The Scheduler interface allows a client to	CPU cores

Object	Description	Relevance for the assets
	manage CPU resources. The API provides functions to query scheduler information, check the online state of CPUs, query CPU idle time and to start threads on defined CPU sets.	
Kernel factory	The kernel factory can be used to create the following types of kernel objects: • Tasks • Threads • Kernel Factory • IPC-Gates • Semaphores • Virtual Machines • DMA spaces • IRQs	Memory, Device, CPU cores, TSF data
IPC-Gate	IPC-Gates are used to create secure communication channels between threads in different compartments. An IPC gate object can be created using the Factory interface. With l4_ipc_gate_bind_thread() a thread is bound to an IPC gate which then receives all messages sent to that IPC gate. The definition of these communication channels is part of the SSP which is determined by the configuration data.	TSF data
IRQ	The IRQ interface provides access to abstract interrupts provided by the microkernel. Interrupts may be • hardware interrupts provided by the platform interrupt controller, • virtual device interrupts provided by the microkernel's virtual devices (e.g., virtual serial or trace buffer) or • virtual interrupts that can be triggered by user programs (IRQs) IRQ objects can be created using a kernel factory.	TSF data, Device
I/O ports (for x86)	Hardware on Intel x86 systems may use	Device

Object	Description	Relevance for the assets
	I/O ports in addition to MMIO. Such hardware can only be accessed if appropriate I/O port capabilities are present in the respective task providing access to I/O ports the hardware is assigned to.	
Semaphore	Using a semaphore, tasks can implement a synchronization operation. This is the interface for kernel-provided semaphore objects. The object provides the classical functions up() and down() for counting the semaphore, where down() blocks as long as the counter is 0.	TSF data, Device
Kernel DMA space	Kernel DMA spaces are used internally in the TOE to manage IOMMU page tables. Applications may create kernel DMA spaces via the kernel factory but cannot use them to manipulate the IOMMU. Applications must use Moe DMA spaces to configure DMA memory.	Device, Memory
VM	VM objects provide access to stage 2 or nested page tables to configure guest physical memory for HW-virtualized guests.	Memory
SMC (for ARM)	The SMC object controls access to the Arm Trustzone.	Memory
Dataspace (Moe object)	Interface for memory-like objects. Dataspaces are a central abstraction provided by L4Re. A dataspace is an abstraction for anything that is available via usual memory access instructions. A dataspace can be a file, as well as the memory-mapped registers of a device, or anonymous memory, such as a heap.	Memory
Dma_space (Moe object)	A Moe-Dma_space represents DMA accessible memory of a DMA capable device. Whenever a device needs direct access to parts of a dataspace, that part of the dataspace must be mapped to the DMA address space that is assigned to that device. A Moe-DMA_space can wrap	Device, Memory

Object	Description	Relevance for the assets
	several kernel DMA spaces.	
Factory (Moe object)	A factory object of Moe can be used to create the following objects in Moe: • Dataspace • Dma_space • Factory (of Moe) • Namespace • Rm • Scheduler • Log	Memory, Device, CPU cores, TSF data
Log (Moe object)	Analogous to the kernel's Vlog object with some convenience functions added, e.g., colors and multiplexing of several clients.	TSF data
Namespace (Moe object)	This is a basic abstraction for managing a mapping from human-readable names to capabilities. In particular, a name can also be mapped to a capability that refers to another name space object. By this means name spaces can be constructed hierarchically.	TSF data
Region_map (Moe object)	The central purpose of the region-map API is to provide means to manage the virtual memory address space of an L4 task. A region-map object implements two protocols. The first protocol is the kernel page-fault protocol, to resolve page faults for threads running in an L4 task. The second protocol is the region-map protocol itself, that allows to attach a dataspace object to a region of the virtual address space.	Memory
Scheduler (Moe object)	The scheduler object of Moe allows a client to manage CPU resources. The API provides functions to query scheduler information, check the online state of CPUs, query CPU idle time and to start threads on defined CPU sets.	CPU cores
L4Re ITAS	The L4Re In-Task Service (ITAS) is the startup code that is run when an L4Re application is started and which serves as	Memory

Object	Description	Relevance for the assets
	pager and exception handler at runtime.	
Objects created based on Ned Lua config (Ned object)	Objects which are created in the Ned Lua config and passed to tasks outside the TOE.	Memory, Device, CPU cores
Vbus as defined in the Io Lua config (Io object)	The virtual bus (Vbus) is a hierarchical (tree) structure of device nodes where each device has a set of resources attached to it. Each virtual bus provides an Icu (Interrupt handler) for interrupt handling. The Vbus interface allows a client to find and query devices present on his virtual bus. After obtaining a device handle for a specific device the client can enumerate its resources. A Vbus as defined in the Io Lua config speaks also the dataspace protocol (for device memory), the inhibitor protocol and the event protocol.	Device, TSF data
Vicu (Io object)	Interface for registering IRQs to device interrupts found on the Vbus.	Device
Platform_control (Io object)	The Platform_control object may be given out in the Ned config to enable to suspend or shutdown outside the TOE.	TSF data

Table 4: Objects

Application Note: This access control policy concerns objects where the access control is managed via memory capabilities, I/O port capabilities, and L4Re object capabilities.

Table 4 only contains objects that can leave the TOE in one of the permitted configurations.

6.1.1.5 FDP_ACF.1/ME Security Attribute based Access Control – Memory

FDP_ACF.1.1/ME The TSF shall enforce the **memory access control policy** to objects based on the following:

- **subject security attributes: compartments with unambiguous reference,**
- **objects security attributes: memory ranges referenced by physical address spaces**.

FDP_ACF.1.2/ME The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **A task in a compartment can only access the memory that is assigned to it**.

FDP_ACF.1.3/ME The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **The kernel-user memory holding the user thread control blocks (UTCBs), the VCPU state, the extended VCPU state and allocated, but not used kernel-user memory is always accessible to every thread of the corresponding task**.

FDP_ACF.1.4/ME The TSF shall explicitly deny access of subjects to objects based on the following additional rules: **none.**

Application Note: Establishment of a shared memory segment between two compartments does not violate capability separation. Assigning non-ECC RAM memory ranges that are physically on the same RAM chip to different compartments is allowed with capability separation.

6.1.1.6 FDP_ACF.1/CC Security Attribute based Access Control – CPU Core

FDP_ACF.1.1/CC The TSF shall enforce the **CPU access control policy** to objects based on the following:

- **subject security attributes: compartments with unambiguous reference,**
- **object security attributes: CPU cores with unambiguous CPU core reference that remains unchanged across reboots**.

FDP_ACF.1.2/CC The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **A thread in a compartment is only scheduled on the assigned CPU cores**.

FDP_ACF.1.3/CC The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **none**.

FDP_ACF.1.4/CC The TSF shall explicitly deny access of subjects to objects based on the following additional rules: **If two compartments c_1 and c_2 shall be strongly separated, then:**

- **c_1 and c_2 have CPUs assigned that are present on different sockets that do not share L1 and L2 caches, branch predictors, and TLB caches.**

Application Note: Capability separation is ensured by assigning CPU cores provided by one CPU socket to different compartments. Assigning CPU hyperthreads provided by one CPU core to different compartments is compatible with capability separation.

6.1.1.7 FDP_ACF.1/DE Security Attribute based Access Control - Device

FDP_ACF.1.1/DE The TSF shall enforce the **device access control policy** to objects based on the following:

- **subjects security attributes: compartments with unambiguous reference,**

- **object security attributes: PCI devices referenced with their PCI device identifier**[4].

FDP_ACF.1.2/DE The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **A task in a compartment is only allowed to access the assigned PCI device referenced by the PCI device identifier**.

FDP_ACF.1.3/DE The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **none**.

FDP_ACF.1.4/DE The TSF shall explicitly deny access of subjects to objects based on the following additional rules: **If two compartments c_1 and c_2 shall be strongly separated, then:**

- **if PCI-ACS is available, then no PCI device may be simultaneously assigned to c_1 and c_2 and PCI-ACS must be configured to prevent direct communication of devices assigned to c_1 with devices assigned to c_2; otherwise**
- **if PCI-ACS is not available, then PCI devices assigned to c_1 and PCI devices assigned to c_2 must not share the same port of the PCI root complex.**

Application Note: A “PCI device” can be function 0 of a single-function, a function of a multi-function (multi-function bit set in type header field) or a virtual function of a virtual-function device (PCI SR-IOV).

If two compartments are each assigned different virtual functions of a virtual-function PCI device, then these two compartments are not strongly separated.

When several independent functions are integrated into a single device, it will be referred to as a multi-function device. Each function on a multi-function device has its own configuration space. If two compartments have different functions of the same multi-function device assigned then these two compartments are strongly separated under the following properties:

- the functions are implemented inside the PCI root complex (root complex integrated devices) or,
- the functions do not share hardware resources that could be used for direct communication.

Otherwise the compartments are not strongly separated. The system integrator has to document the assumptions which imply strong separation if independent functions of a multi-function device are assigned to different compartments.

On the LX2160A platform, PCI devices are not supported by L4Re SSK.

[4] The device identifier is defined hereafter by the tuple Bus:Device.Function, where Bus is the PCI Bus number, Device the PCI Device number, and Function the PCI Function number.

6.1.1.8 FDP_ACF.1/OB Security Attribute based Access Control – Objects

FDP_ACF.1.1/OB The TSF shall enforce the **objects access control policy** to objects based on the following:

- **subjects security attributes: capability present in the capability array of the calling task belonging to a compartment,**
- **object security attribute: capability to which the object of Table 4 points, permissions of the capability**

FDP_ACF.1.2/OB The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **If a subject owns the capability, it can communicate with the object the capability points to. The type of communication depends on the permissions of the capability.**

FDP_ACF.1.3/OB The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **none**.

FDP_ACF.1.4/OB The TSF shall explicitly deny access of subjects to objects based on the following additional rules: **none**.

Application Note: The allowed operations to be exercised on a capability depend on the type of the capability, i.e. which object the capability represents, and the capability's permissions. During IPC, the permissions can be transferred from the sender to receiver along with the capability depending on the wants of the sender and the permissions of the channel.

For memory capabilities, the available permissions are read, write and execute as far as the underlying hardware supports those. IO port capabilities only have an implicit combined read-write permission.

Five different permissions for L4Re object capabilities are supported:

- read: Provides access to the capability. Implicit permission; if the read permission is revoked, the capability is removed.
- write: Object specific permission
- special: Access permission that guards object creation in factories and kernel internal links that could prevent object deletion.
- delete: Permission for object deletion.
- server: Permission for the server functionality of IPC gates

In addition, there exists the weak capability, which is not really a traditional permission, but behaves like one. The L4Re Microkernel deletes the referenced object when the last non-weak capability is deleted or unmapped.

6.1.1.9 FDP_IFC.2 Complete Information Flow Control

FDP_IFC.2.1 The TSF shall enforce the **capability information flow control policy** on

- **subjects: compartments,**

- **information: capability referencing an object as of Table 4 controlled by the TOE, information transferred with operation on capability**

and all operations that cause that information to flow to and from subjects covered by the SFP.

FDP_IFC.2.2 The TSF shall ensure that all operations that cause any information in the TOE to flow to and from any subject in the TOE are covered by an information flow control SFP.

6.1.1.10 FDP_IFF.1 Simple Security Attributes

FDP_IFF.1.1 The TSF shall enforce the **capability information flow control policy** based on the following types of subject and information security attributes:

- **subject security attributes: compartments with unambiguous reference,**
- **information security attributes: reference to passive subject/object reference the capability applies to, assignment of capability to active subject allowed to perform operations on capability, operations defined for capability**.

FDP_IFF.1.2 The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold:

- **the active subject possessing a capability is able to perform the defined operations on the capability to transfer data to/from the passive subject/object the capability applies to,**
- **a capability on a passive subject allows the active subject to transfer a copy of a capability in its possession to that passive subject.**

FDP_IFF.1.3 The TSF shall enforce the **following additional rules: The TOE allows the definition of a data channel between subjects that denies the transmission of capabilities**.

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: **none**.

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: **none**.

6.1.1.11 FDP_RIP.1 Subset residual information protection

FDP_RIP.1.1 The TSF shall ensure that any previous information content of a resource is made unavailable upon the **allocation of the resource to** the following objects:

- **general purpose CPU registers,**
- **floating point CPU registers,**

- **mathematical extension CPU registers pertaining to SSSE3, AVX, AVX2,**
- **TLB cache lines,**
- **CPU branch predictor,**
- **memory**

6.1.2 Security Management (FMT)

6.1.2.1 FMT_MSA.3/ME Static Attribute Initialisation – Memory

FMT_MSA.3.1/ME The TSF shall enforce the **memory access control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/ME The TSF shall allow ~~the~~ **nobody** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.2 FMT_MSA.3/CC Static Attribute Initialisation – CPU Core

FMT_MSA.3.1/CC The TSF shall enforce the **CPU access control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/CC The TSF shall allow ~~the~~ **nobody** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.3 FMT_MSA.3/DE Static Attribute Initialisation – Device

FMT_MSA.3.1/DE The TSF shall enforce the **device access control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/DE The TSF shall allow ~~the~~ **nobody** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.4 FMT_MSA.3/OB Static Attribute Initialisation – Objects

FMT_MSA.3.1/OB The TSF shall enforce the **objects access control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/OB The TSF shall allow ~~the~~ **nobody** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.5 FMT_MSA.3/CAP Static Attribute Initialisation – Capability Information Flow Control

FMT_MSA.3.1/CAP The TSF shall enforce the **capability information flow control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/CAP The TSF shall allow ~~the~~ **nobody** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.6 FMT_MTD.1/CAP Management of TSF Data – Capability

FMT_MTD.1.1/CAP The TSF shall restrict the ability to **provide** the **capabilities** to the **task possessing the capability**.

6.1.2.7 FMT_SMF.1 Specification of Management Functions

FMT_SMF.1.1 The TSF shall be capable of performing the following management functions:

- **management of the capability information flow control**.

6.1.3 Identification and Authentication (FIA)

6.1.3.1 FIA_UID.2 User Identification

FIA_UID.2.1 The TSF shall require each ~~**user**~~ ***compartment*** to be successfully identified before allowing any other TSF-mediated actions on behalf of that ~~**user**~~ ***compartment***.

6.1.4 Privacy (FPR)

6.1.4.1 FPR_UNO.1 Unobservability

FPR_UNO.1.1 The TSF shall ensure that **compartments without access to the following objects** are unable to observe the operation **all operations** on

- **memory,**
- **PCI devices,**
- **CPU resources,**
- **objects of Table 4**

by **compartments with access to these objects**.

6.2 Rationale for Security Functional Requirements

6.2.1 Coverage

The following table provides a mapping of the SFRs to the security objectives, showing that each security functional requirement addresses at least one security objective.

Security functional requirements	Objectives
FDP_ACC.2/ME	O.CONFIDENTIALITY O.INTEGRITY O.AVAILABILITY
FDP_ACC.2/CC	O.INTEGRITY
FDP_ACC.2/DE	O.CONFIDENTIALITY O.INTEGRITY

Security functional requirements	Objectives
	O.AVAILABILITY
FDP_ACC.2/OB	O.CONFIDENTIALITY O.INTEGRITY O.AVAILABILITY
FDP_ACF.1/ME	O.CONFIDENTIALITY O.INTEGRITY O.AVAILABILITY
FDP_ACF.1/CC	O.INTEGRITY
FDP_ACF.1/DE	O.CONFIDENTIALITY O.INTEGRITY O.AVAILABILITY
FDP_ACF.1/OB	O.CONFIDENTIALITY O.INTEGRITY O.AVAILABILITY
FDP_IFC.2	O.CONFIDENTIALITY
FDP_IFF.1	O.CONFIDENTIALITY
FDP_RIP.1	O.CONFIDENTIALITY
FMT_MSA.3/ME	O.CONFIDENTIALITY O.INTEGRITY
FMT_MSA.3/CC	O.CONFIDENTIALITY O.INTEGRITY
FMT_MSA.3/DE	O.CONFIDENTIALITY O.INTEGRITY
FMT_MSA.3/OB	O.CONFIDENTIALITY O.INTEGRITY
FMT_MSA.3/CAP	O.CONFIDENTIALITY O.INTEGRITY
FMT_MTD.1/CAP	O.AVAILABILITY
FMT_SMF.1	O.AVAILABILITY
FIA_UID.2	O.CONFIDENTIALITY O.INTEGRITY

Security functional requirements	Objectives
FPR_UNO.1	O.CONFIDENTIALITY

Table 5: Mapping of SFRs to Security Objectives

6.2.2 Sufficiency

The following table provides justification for each security objective for the TOE, showing that the security functional requirements are suitable to meet and achieve the security objectives.

Security Objectives for the TOE	Rationale
O.CONFIDENTIALITY	The memory access control policy defined in FDP_ACC.2/ME and FDP_ACF.1/ME makes the memory non-read-accessible to non-authorized entities. The device access control policy defined in FDP_ACC.2/DE and FDP_ACF.1/DE makes devices non-accessible to non-authorized entities. The objects access control policy defined in FDP_ACC.2/OB and FDP_ACF.1/OB makes objects of Table 4 non-accessible to non-authorized entities. The clearing of shared resources upon reallocation ensures the confidentiality of data as claimed by FDP_RIP.1. The SFRs FMT_MSA.3/ME, FMT_MSA.3/CC, FMT_MSA.3/DE, FMT_MSA.3/OB, FMT_MSA.3/CAP, contribute in fulfilling this objective by providing management functions for the attributes associated to the respective policy. FIA_UID.2 ensures that compartments are identified. FDP_IFF.1, FDP_IFC.2 ensure that capability information flows are governed with the capability system offered by the TOE. FPR_UNO.1 ensures that compartments without access to objects maintained by the TOE (except CPU cores) cannot observe the operation on these objects by compartments having access.
O.INTEGRITY	The memory access control policy defined in FDP_ACC.2/ME and FDP_ACF.1/ME makes the memory non-read-accessible to non-authorized entities. The CPU core access control policy defined in FDP_ACC.2/CC and FDP_ACF.1/CC makes the CPU cores only accessible to authorized entities. The device access control policy defined in FDP_ACC.2/DE and FDP_ACF.1/DE makes devices non-accessible to non-authorized entities. The objects access control policy defined in FDP_ACC.2/OB and FDP_ACF.1/OB makes objects of Table 4 non-accessible to non-authorized entities. The SFRs FMT_MSA.3/ME, FMT_MSA.3/CC, FMT_MSA.3/DE, FMT_MSA.3/OB, FMT_MSA.3/CAP, contribute in fulfilling this objective by providing management functions for the attributes associated to the respective policy. FIA_UID.2 ensures that compartments are identified.
O.AVAILABILITY	This objective is addressed as the access to the memory is controlled by FDP_ACC.2/ME and FDP_ACF.1/ME and then no

Security Objectives for the TOE	Rationale
	uncontrolled/unauthorized access can be performed over the memory for making them unavailable for the compartments. This objective is addressed as the access to the devices is controlled by FDP_ACC.2/DE and FDP_ACF.1/DE and then no uncontrolled/unauthorized access can be performed over the device for making them unavailable for the compartments. This objective is addressed as the access to the objects is controlled by FDP_ACC.2/OB and FDP_ACF.1/OB and then no uncontrolled/unauthorized access can be performed over the objects of Table 4 for making them unavailable for the compartments. The SFR FMT_MTD.1/CAP restricts the provisioning of capabilities to tasks possessing the capability. FMT_SMF.1 provides management functionality to ensure that tasks and threads are available.

Table 6: SFR Sufficiency Analysis

6.2.3 SFR Dependencies

The following table provides the SFR dependency analysis.

SFR	Dependencies	Dependency Coverage in ST
FDP_ACC.2/ME	FDP_ACF.1	Yes, by FDP_ACF.1/ME
FDP_ACC.2/CC	FDP_ACF.1	Yes, by FDP_ACF.1/CC
FDP_ACC.2/DE	FDP_ACF.1	Yes, by FDP_ACF.1/DE
FDP_ACC.2/OB	FDP_ACF.1	Yes, by FDP_ACF.1/OB
FDP_ACF.1/ME	FDP_ACC.1 FMT_MSA.3	Yes, by FDP_ACC.2/ME (hierarchical) Yes, by FMT_MSA.3/ME
FDP_ACF.1/CC	FDP_ACC.1 FMT_MSA.3	Yes, by FDP_ACC.2/CC (hierarchical) Yes, by FMT_MSA.3/CC
FDP_ACF.1/DE	FDP_ACC.1 FMT_MSA.3	Yes, by FDP_ACC.2/DE (hierarchical) Yes, by FMT_MSA.3/DE
FDP_ACF.1/OB	FDP_ACC.1 FMT_MSA.3	Yes, by FDP_ACC.2/OB (hierarchical) Yes, by FMT_MSA.3/OB
FDP_IFC.2	FDP_IFF.1	Yes
FDP_IFF.1	FDP_IFC.1	Yes, by FDP_IFC.2 (hierarchical)

SFR	Dependencies	Dependency Coverage in ST
	FMT_MSA.3	Yes
FDP_RIP.1	No dependencies	No dependencies
FMT_MSA.3/ME	FMT_MSA.1 FMT_SMR.1	FMT_MSA.1: No. This SFR not applicable, as the TSF does not protect its management. This is also not required considering that nobody can change the default values. FMT_SMR.1: No. As no management of the default value is defined, the dependency to FMT_SMR.1 is unresolved.
FMT_MSA.3/CC	FMT_MSA.1 FMT_SMR.1	FMT_MSA.1: No. This SFR not applicable, as the TSF does not protect its management. This is also not required considering that nobody can change the default values. FMT_SMR.1: No. As no management of the default value is defined, the dependency to FMT_SMR.1 is unresolved.
FMT_MSA.3/DE	FMT_MSA.1 FMT_SMR.1	FMT_MSA.1: No. This SFR not applicable, as the TSF does not protect its management. This is also not required considering that nobody can change the default values. FMT_SMR.1: No. As no management of the default value is defined, the dependency to FMT_SMR.1 is unresolved.
FMT_MSA.3/OB	FMT_MSA.1 FMT_SMR.1	FMT_MSA.1: No. This SFR not applicable, as the TSF does not protect its management. This is also not required considering that nobody can change the default values. FMT_SMR.1: No. As no management of the default value is defined, the dependency to FMT_SMR.1 is unresolved.
FMT_MSA.3/CAP	FMT_MSA.1 FMT_SMR.1	FMT_MSA.1: No. This SFR not applicable, as the TSF does not protect its management. This is also not required considering that nobody can change the default values. FMT_SMR.1: No. As no management of the default value is defined, the dependency to FMT_SMR.1 is unresolved.
FMT_MTD.1/CAP	FMT_SMR.1 FMT_SMF.1	FMT_SMR.1: No. The management of capabilities is made possible when the active subject possesses a capability. Using the capability

SFR	Dependencies	Dependency Coverage in ST
		mechanism defined with FDP_IFC.2 and FDP_IFF.1, the management of capabilities is controlled. FMT_SMF.1: Yes
FMT_SMF.1	No dependencies.	No dependencies.
FIA_UID.2	No dependencies.	No dependencies.
FPR_UNO.1	No dependencies.	No dependencies

Table 7: SFR Dependencies

6.3 Security Assurance Requirements

The security assurance requirements (SARs) for the TOE are the Evaluation Assurance Level 4 components, augmented by ALC_FLR.3, as specified in [CC] Part 3. No operations are applied to any of the assurance components.

These components are listed in Table 8.

Security assurance class	Security assurance requirement	Source
ADV Development	ADV_ARC.1 Security architecture description	[CC] Part 3
	ADV_FSP.4 Complete functional specification	[CC] Part 3
	ADV_IMP.1 Implementation representation of the TSF	[CC] Part 3
	ADV_TDS.3 Basic modular design	[CC] Part 3
AGD Guidance documents	AGD_OPE.1 Operational user guidance	[CC] Part 3
	AGD_PRE.1 Preparative procedures	[CC] Part 3
ALC Life-cycle support	ALC_CMC.4 Production support, acceptance procedures and automation	[CC] Part 3
	ALC_CMS.4 Problem tracking CM coverage	[CC] Part 3
	ALC_DEL.1 Delivery procedures	[CC] Part 3
	ALC_DVS.1 Identification of security measures	[CC] Part 3
	ALC_FLR.3 Systematic flaw remediation	[CC] Part 3
	ALC_LCD.1 Developer defined life-cycle model	[CC] Part 3
	ALC_TAT.1 Well-defined development tools	[CC] Part 3
ASE Security Target evaluation	ASE_CCL.1 Conformance claims	[CC] Part 3
	ASE_ECD.1 Extended components definition	[CC] Part 3
	ASE_INT.1 ST introduction	[CC] Part 3
	ASE_OBJ.2 Security objectives	[CC] Part 3

	ASE_REQ.2 Derived security requirements	[CC] Part 3
	ASE_SPD.1 Security problem definition	[CC] Part 3
	ASE_TSS.1 TOE summary specification	[CC] Part 3
ATE Tests	ATE_COV.2 Analysis of coverage	[CC] Part 3
	ATE_DPT.1 Testing: basic design	[CC] Part 3
	ATE_FUN.1 Functional testing	[CC] Part 3
	ATE_IND.2 Independent testing - sample	[CC] Part 3
AVA Vulnerability Assessment	AVA_VAN.3 Focused vulnerability analysis	[CC] Part 3

Table 8: Security Assurance Requirements

6.4 Security Assurance Requirements Rationale

EAL 4 has been considered appropriate for the threats defined in the security problem definition.

The augmentation with ALC_FLR.3 has been chosen to provide greater assurance regarding the developer’s continuous flaw remediation processes.

6.5 Security Assurance Requirements Dependency Analysis

The set of SARs included in this ST is the one associated to the EAL4 assurance package, whose internal dependencies are satisfied, augmented by ALC_FLR.3.

ALC_FLR.3 depends on: No dependencies.

Therefore, all the dependencies for the selected set of SARs are satisfied.

7 TOE Summary Specification

The following section explains how the security functions in this security target are implemented. The different TOE and product platform security functions cover the various SFRs listed in the previous section. The security functions address the objects given in Table 4.

7.1 Separation of Compartments

The microkernel-based L4Re Operating System Framework, of which the product that contains the TOE is a distribution, is capable of exclusively assigning physical as well as virtual resources to the compartments. Such exclusively assigned resources cannot be used as communication channel between compartments. Communication between compartments is only possible using shared resources dedicated to this purpose. The initial resource assignment is statically defined during boot-time based on the configuration. Additional data and capability channels can be established via the already existing capability channels. These newly created channels will not alter the boot-time defined communication matrix, i.e. compartments that are allowed to exchange capabilities can establish additional data and capability channels with each other. Compartments that initially do not have any capability channel defined have no ability to directly establish a data or capability channel with each other at run-time. Such compartments can only use the data channels defined in the initial configuration, if there are any. Because communication between compartments is only possible using the defined data and capability channels, capability separation as well as strong separation can be achieved for any pair of compartments.

Capability separation of the compartments allows the communication via well-defined data channels established according to the boot-time defined communication matrix. Compartments that shall be strongly separated from each other have to be capability-separated and must neither possess a direct data channel nor share a device. If configured at boot-time, such compartments may exchange data only via a proxy, in case the proxy permits this. The NVMe Server, for instance, does not permit any communication between its clients, thus enforcing the separation between them. Note that timing side channels and covert channels via caches or other shared CPU resources are not addressed by the capability separation or strong separation. If desired, the initial configuration can enforce the execution of different compartments on different CPUs or dies to limit such channels.

The TOE provides two communication proxies, which may be used to connect separated compartments for certain application scenarios. Additionally, the evaluation includes a developer guide documenting all necessary constraints to be observed by developers when implementing a proxy with a protocol that is considered to be suitable for the business logic of the compartments. Although the guide documenting such development constraints is part of the evaluation, the proxies developed based on this guide are not covered by this evaluation and its results. Using this developer guide, one may, for instance, develop a proxy that passes data only in one direction for a data diode based on the TOE.

Compartments are assigned resources during boot time. Resources can be passed to other compartments at runtime only via capability channels. Each assigned resource remains with the respective compartments for the lifetime of the system, with the possible exception of memory.[5]

[5] Memory is managed by dataspace objects, which are created, owned and managed by Moe. A compartment can destroy a dataspace during runtime in order to give up its assigned memory in that dataspace. In such a case, Moe assumes control over the memory pages in that dataspace and sanitizes the memory before assigning it to other compartments.

The resource assignment includes exclusive access to resources, such as PCI devices, as well as access to resources shared with other compartments, such as memory.

Resources with exclusive access can be utilized by compartments without interference from the TOE after the assignment during boot. Access to shared resources, on the other hand, is controlled by the TOE to ensure domain separation of the compartments from each other. This includes the clearing of residual information upon reallocation of a shared resource to another compartment, such as CPU registers, TLB cache lines, memory, and the CPU branch predictor. PCI- and non-PCI-devices can be reassigned only by booting the TOE with a different configuration. In such cases the integrator must ensure to properly reset the devices to their initial state according to the rules listed below.

The resource management is implemented by the TOE by ensuring that all resources of the hardware platform as well as TOE resources are maintained by the TOE. This covers all hardware resources visible to the TOE and its controlled entities as well as all TOE software objects such as threads, memory objects, communication channels, memory allocators. The following hardware resources are controlled by the TOE and can be assigned to compartments:

- (RAM) memory
- CPU resources, such as cores and time
- PCI devices under the following condition:
  - For PCI devices that permit persistent firmware updates, the integrator must ensure that they are either always assigned to the same compartment or reassigned only to a compartment that is not strongly separated from the previously assigned compartment. There are no restriction on non-persistent firmware updates.
  - No firmware is loaded before the TOE has started (e.g. during device enumeration by the BIOS).
- Non-PCI-devices, if they fulfil the following properties:
  - MMIO ranges must not share a physical page with another device.
  - I/O ports are not shared with another device.
  - Devices for which device-to-device communication cannot be ruled out may establish a data channel and must thus always be assigned to compartments that are not strongly separated. For any two devices assigned to strongly separated compartments direct device-to-device communication must be impossible.
  - If the device permits persistent firmware updates, the integrator must ensure that the device is either always assigned to the same compartment or reassigned only to a compartment that is not strongly separated from the previously assigned compartment. There are no restriction on non-persistent firmware updates.
  - No firmware is loaded before the TOE has started (e.g. during device enumeration by the BIOS).

Note that on the LX2160A platform, PCI devices are not supported by L4Re SSK.

Assigning CPU cores to different compartments is allowed with strong separation. Assigning a CPU core's hardware threads to different compartments is compatible with strong separation. It is the responsibility of the configurator to judge whether the resulting covert channels can be tolerated in

the end product. If not, it is the responsibility of the configurator to create a configuration where the covert channels can be tolerated.

Once the TOE assigns a PCI device to a compartment, this compartment has complete and full control over that PCI device. To guarantee that this device access cannot be abused to violate the security policy of the TOE, the following mechanisms are defined:

- If PCI-ACS is available, then the TOE uses PCI-ACS to ensure that a PCI device does not have unwanted access to other PCI devices. PCI-ACS can be configured to prevent the direct communication between PCI devices that share the same port of the PCI root complex. Hence, it is possible to assign different PCI devices to different strongly separated compartments, even if the PCI devices share the same port of the PCI root complex.
- If PCI-ACS is not available, then PCI devices that share the same port of the PCI root complex may only be assigned to one compartment in each pair of compartments that shall be strongly separated.
- The BIOS of the underlying execution environment must ensure that during PCI device enumeration, no PCI device BIOS is started in privileged mode.

In addition, the TOE allows the assignment of the objects listed in Table 4 to compartments.

The memory of the system is subdivided to form dataspace objects. The kernel-user memory holding the user thread control blocks (UTCBs), the VCPU state, the extended VCPU state and allocated, but not used kernel-user memory is always accessible to every thread of the task for which this kernel-user memory was allocated. Except for kernel-user memory, no kernel memory is shared with deprivileged tasks. Establishment of a shared memory segment between two compartments does not violate capability separation. Assigning non-ECC RAM memory ranges that are physically on the same RAM chip to different compartments is allowed with capability separation.

Scheduling in L4Re SSK is always preemptive and core-local and uses a fixed-priority round-robin algorithm. On a given core, the scheduler always selects the thread with highest priority. Threads of equal priority are scheduled round-robin. Therefore, integrators can always guarantee that critical threads cannot be starved by assigning an appropriate priority to them.

7.2 Information Flow Control

The TOE implements a strict object-oriented structure. All hardware as well as software resources are modelled as objects and can only be referenced via capabilities, which are L4Re SSK managed, unforgeable tokens of authority containing access rights. L4Re tasks including compartments perform operations on these objects by invoking a system call referencing the capability and the operation to be performed with the respective object. Capabilities are only valid within the security domain (i.e., an application) they were mapped to by the microkernel.

Capabilities can be transferred to other security domains via capability channels, which are represented by well-defined interfaces offered by the TOE. The configurator can make the transfer of capabilities between any pair of compartments impossible by choosing an initial configuration, in which these compartments are not connected by a capability channel. In typical configurations capability channels between compartments exist only when one of the two connected compartments is trusted. For instance, clients of virtio servers are connected with the server with a capability channel, because the virtio protocol requires the exchange of certain capabilities.

7.3 System Management

The TOE configuration is defined with several Lua scripts. The configuration allows the definition of either hardware-resource-backed or software objects. In addition, the configuration defines the subjects, that is, the compartments, and assigns the created objects to these subjects.

The Lua scripts are interpreted by the Ned and Io components of the TOE during boot-time, which ensures that the defined objects and subjects are created and that the assignment of objects to subjects is configured. The assignment is achieved by providing the capability of the object to the subject that shall be able to interact with the object.

The TOE does not provide management interfaces that allow system configuration at run-time. The allocation of resources shared with other non-TOE compartments or entities is determined by the Lua script at boot-time and not modifiable at run-time. (Remark: In principle L4Re supports any dynamic reconfiguration. However, compartments are defined here as the initial applications created by ned. Therefore, one cannot create an additional compartment at runtime or create a capability channel between two capability-separated compartments.)

A compartment can only get access to a particular device, if this is permitted in the initial configuration, i.e., if the device is assigned to the compartment. The rules in the preceding section allow this only if the device cannot violate any properties claimed in this ST, regardless of the configuration of the device. Compartments are therefore free to change the configuration of the devices they got assigned.

L4Re SSK supports secure boot on certain x86 platforms, however, this feature is not claimed in this ST. The environment of the TOE has to ensure that the TOE installation is protected against unauthorized modifications. The integrator must follow processes for installation and version update that guarantee the integrity of the installation and prevent version rollback. The Lua scripts and other configuration data are part of the boot image and are therefore integrity protected together with the installation.

7.4 SFR to TSS References

The following table describes how each TOE security function covers the SFRs.

SFR	Coverage in TSS
FDP_ACC.2/ME	Separation of Compartments documented in Section 7.1.
FDP_ACC.2/CC	Separation of Compartments documented in Section 7.1.
FDP_ACC.2/DE	Separation of Compartments documented in Section 7.1.
FDP_ACC.2/OB	Separation of Compartments documented in Section 7.1.
FDP_ACF.1/ME	Separation of Compartments documented in Section 7.1.
FDP_ACF.1/CC	Separation of Compartments documented in Section 7.1.
FDP_ACF.1/DE	Separation of Compartments documented in Section 7.1.
FDP_ACF.1/OB	Separation of Compartments documented in Section 7.1.
FDP_IFC.2	Information Flow Control documented in Section 7.2.

SFR	Coverage in TSS
FDP_IFF.1	Information Flow Control documented in Sections 7.1 and 7.2.
FDP_RIP.1	Separation of Compartments documented in Section 7.1.
FMT_MSA.3/ME	Separation of Compartments documented in Sections 7.1 and 7.3.
FMT_MSA.3/CC	Separation of Compartments documented in Sections 7.1 and 7.3.
FMT_MSA.3/DE	Separation of Compartments documented in Sections 7.1 and 7.3.
FMT_MSA.3/OB	Separation of Compartments documented in Sections 7.1 and 7.3.
FMT_MSA.3/CAP	Information Flow Control documented in Sections 7.2 and 7.3.
FMT_MTD.1/CAP	Security management is documented in Section 7.3.
FMT_SMF.1	Security management is documented in Section 7.3.
FIA_UID.2	Supports the Separation of Compartments documented in Section 7.1 as well as Information Flow Control documented in Section 7.2.
FPR_UNO.1	Separation of Compartments documented in Section 7.1.

Table 9: SFR to TSS References

8 Terms and Definitions

ACS:

Access Control Services (ACS) is a set of control and capability registers used to implement access control over routing within a PCI Express component. The here-mentioned capabilities refer to different types of access control.

Address space:

An *address space* defines a range of discrete addresses.

Address-space separation:

Address-space separation is the default minimal separation of applications. Except for explicitly configured shared memory, each application has access to its private memory only. Each application has its own (private) capability space, whose integrity is protected by the TOE. Limited by access control, applications can share their own capabilities with other applications. Each application can effectively protect itself against receiving capabilities or overwriting slots in its capability space when being granted access rights.

Application:

An *application* is associated to an executable binary. Generally, an application can be a stand-alone one, compiled from a system programming language or an OS called “guest operating system”. A complex L4Re application may consist of several tasks that each contains several threads. Applications are assigned to compartments. The build of such applications binaries and any guidance how they are build are not part of the TOE except for:

- warnings in the TOE guidance that capabilities shall not be shared
- the guidance how to create a communication proxy

For further information on the build of applications please contact Kernkonzept.

Bootloader:

see “Firmware”.

Capability:

A *capability* is a tuple consisting of a reference to an object of Table 4 and permissions. Applications can only invoke or use features of objects for which they posses a suitable capability. All system calls contain at least one capability argument, which names the target of this system call. The available features depend on the permissions, for instance, a memory capability without the write permission only provides read only access to that memory.

For memory capabilities, the available permissions are read, write and execute as far as the underlying hardware supports those. IO port capabilities only have an implicit combined read-write permission.

Five different permissions for capabilities are supported:

- read: Provides access to the capability. Implicit permission; if the read permission is revoked, the capability is removed.
- write: Object specific permission
- special: Access permission that guards object creation in factories and kernel internal links that could prevent object deletion.
- delete: Permission for object deletion.
- server: Permission for the server functionality of IPC gates

In addition, there exists the weak capability, which is not really a traditional permission, but behaves like one. The L4Re Microkernel deletes the referenced object when the last non-weak capability is deleted or unmapped.

Capabilities are stored in capability spaces inside tasks in kernel-protected memory. Capabilities can be transferred to other tasks, which means that the receiving task obtains a new capability referring to the same object, possibly with reduced permissions.

Capability channel:

A *capability channel* is a communication channel that allows the transfer of capabilities from the sending task to the receiving task. A capability channel is always based on a task, thread, or IPC gate capability.

Capability mediation:

Capability mediation is a property whereby the TSF provides the possibility to transfer capabilities via capability channels.

Capability separation:

Two compartments are *capability-separated* if they have no means to exchange capabilities; neither directly nor with the help of other compartments. Compartments with capability separation therefore allow the exchange of user data via system integrator defined data channels. These data communication channels cover the types listed below. Note, some of these communication channels may offer separation functionality (like the SR-IOV functionality listed below). However, those separation mechanisms are not controlled and enforced by the TOE, which implies that this ST cannot claim that they are effective. Hence, they are viewed as potentially allowing communication without being prevented by the TOE.

While capability separation forbids exchange of capabilities, it still allows for example the following data channels:

- Establishment of a shared memory segment between two compartments.

- Assigning different PCI devices attached to a PCIe bus part that does not traverse the PCI root complex to different compartments.
- Assigning different PCI functions provided by one PCI device to different compartments. As an extension, this statement also covers SR-IOV devices where the different PCI device functions are separated by the SR-IOV functionality enforced by the PCI device itself.
- Assigning physical CPU cores to different compartments.
- Assigning CPU hyperthreads provided by one CPU core to different compartments.
- Assigning non-ECC RAM memory ranges which are physically on the same RAM chip to different compartments.

Capability space:

The *capability space* contains all capabilities that are accessible to the threads of this task, including capabilities for memory, I/O ports, memory mapped devices, and capabilities referring to software objects, such as dataspaces.

Communication channel:

A *communication channel* may exist between two tasks and is based on a shared resource. For example, a piece of memory can be used for a communication channel if two tasks have appropriate memory capabilities in order to read/write that piece of memory. A communication channel can be a capability as well as a data channel.

Compartment:

The collection of applications consisting of one initial application together with all applications started by this initial application. Typically, an initial application does not start additional applications.

Configuration data:

The *configuration data* defines a set of rules on how the TOE behaves. The default configuration is the valid start configuration. Any communication between compartments has to be explicitly allowed by the integrator in the configuration data. See also System Security Policy.

Covert channel:

A *covert channel* is a communication channel that allows a process to transfer information in a manner that violates the system’s security policy.

(Bi-/Unidirectional) data channel:

Bidirectional data channels are shared memory, semaphores and IRQs which allow to transfer data and signals in both directions but do not permit the transfer of capabilities. As a special case, shared access to a physical device is considered as a bidirectional data channel in this document.

Unidirectional data channels are semaphores and IRQs with restricted permissions which do only permit to trigger signals but not to receive signals or to transfer capabilities.

Firmware:

Firmware is a hardware-specific software which is written in the non-volatile memory of the hardware that initializes the hardware after the power on and that (fully or partially) loads the TOE into RAM memory and hands over the full control to the TOE. Secure boot has to be implemented by the bootloader (see Section 1.3.5).

Hardware:

Hardware is the physical part of the TOE operational environment on which the TOE is executed. Usually, hardware is a board with several components such as CPUs and I/O devices (e.g. serial interfaces, network adapters) etc. This ST considers firmware as part of the hardware.

Integrator:

The *integrator* is a user who has access to the Lua scripts while L4Re SSK is non-operational, i.e. when it is offline. Typically, the integrator either has physical access to the hardware to access the Lua scripts or is in another way given access to the boot image or boot partition. Since L4Re SSK is non-operational whenever the integrator accesses the Lua scripts, L4Re SSK does not control the role of the integrator. The integrator is in particular responsible for the configuration of the system. For the use case of a communication proxy, the system integrator must ensure that a particular proxy implementation is configured as intended by the developer.

IOMMU:

By an *input/output memory management unit* (IOMMU) device-visible virtual addresses are mapped to physical addresses. IOMMU allows the use of DMA-capable hardware in virtual environments by DMA or interrupt remapping.

IPC:

IPC is a communication protocol to exchange messages within a compartment or between compartments synchronously. The communication objects are IPC messages.

IPC-Gate:

IPC-Gates are used to create secure communication channels between threads in different compartments.

IRQ:

An *interrupt request* (IRQ) is an interrupt which may be a hardware interrupt provided by the platform interrupt controller, a virtual device interrupt provided by the microkernel's virtual devices (virtual serial or trace buffer) or a virtual interrupt that can be triggered by user programs (IRQs).

L4Re micro hypervisor:

see "L4 microkernel".

L4Re object:

L4Re objects referenced by capabilities are the ones listed in Table 4 except memory pages and I/O ports. A given object can be referenced by one or more capabilities, but each capability references exactly one object.

L4 microkernel:

The *L4 microkernel* (also called Fiasco.OC microkernel) is here used as a hypervisor, the L4Re micro hypervisor.

L4Re In-Task Service:

The startup code that is run when an L4Re application is started and which serves as pager and exception handler at runtime.

PCIe branch:

A *PCIe branch* is a branch starting in the same PCI port of the PCI root complex.

PCI root complex:

A *PCI root complex* connects the CPU and memory subsystem to the PCI device. It may support several PCI ports. Each PCI port is connected to an endpoint PCI device or else to a switch that then forms a sub-hierarchy interconnecting many PCI devices. The root complex transmits packets out of its ports and also receives packets into its ports which it then forwards to memory or the CPU.

(Communication) Proxy:

A *(communication) proxy* is a compartment that allows two or more other compartments to set up communication channels while ensuring certain separation properties. Note that a proxy can only maintain separation properties between compartments but cannot enforce them. Therefore, it is crucial that compartments that shall be separated are actually separated after the startup of the system. This has to be ensured by the system integrator.

Semaphore:

A *semaphore* is an abstract data type used to control the access to a common resource by multiple tasks. Using a semaphore, tasks can implement a synchronization operation.

SMMU:

The ARM *System Memory Management Unit* (SMMU) specification outlines an IOMMU architecture for ARM processors.

Strong separation:

We call two compartments *strongly separated* if they are capability-separated and cannot exchange information without passing it through a proxy – if configured at boot-time in accordance with the guidance. Note that a trusted compartment may have the capabilities to cause the exchange of capabilities/information between capability-separated/strongly separated compartments; however, it is assumed that a trusted compartment does not actually do that without intention of the system integrator. A further exception are trusted multi-function PCI devices where different functions of that device can be assigned to different strongly separated compartments while these compartments retain their strong separation.

Note that timing side channels are not addressed by the capability separation or strong separation.

Subject:

In this ST the term *subject* is used for a thread in a compartment, for a task, or for a compartment as a whole depending on the context.

System Security Policy (SSP):

The configuration data uniquely defines the *System Security Policy* consisting of the configuration choices made by the integrator regarding resources and devices and their allocation to compartments. The SSP defines compartments, sets their resources and devices, and defines the allowed communication channels. The TOE configuration enforcing the SSP is defined by means of a Lua script.

Task:

A *task* is a tuple consisting of a set of threads and a capability space. The associated threads are those ones that run “inside” this task. The capability space contains all capabilities that are accessible to the threads of this task, including capabilities for memory, I/O ports, memory mapped devices, and capabilities referring to software objects, such as dataspaces.

Thread:

A *thread* is an executable piece of code including the necessary control information, such as instruction pointer, stack pointer, and other CPU register content. In a compartment there can be multiple threads and each thread is uniquely assigned to its compartment.

Trusted/Untrusted compartment:

A compartment is called *trusted* if it is ensured, e.g., by implementing an appropriate specification, that the compartment does not inadvertently, i.e., without intention of the integrator, invalidate certain separation properties. A compartment that is not trusted is called *untrusted*. In particular, each instance of a proxy constitutes its own untrusted compartment.

Trusted multi-function PCI device:

A *trusted multi-function PCI device* is e.g. a multi-function PCI device whose implementation can be inspected or which has been authorized by a trusted authority.

User Thread Control Block (UTCB):

The *User Thread Control Block (UTCB)* is a data structure defined by the L4 microkernel and located on kernel-provided memory. Each thread gets a unique UTCB assigned when it is bound to a task.

VirtIO-Net:

VirtIO-Net is the para-virtualization solution used for networking.

VT-x:

Intel®'s technology for virtualization on the x86 platform.

9 Abbreviations

Abbreviation	Description
ACS	Access Control Services
AMT/ME	Active Management Technology/Management Engine
API	Application Programming Interface
AVX	Advanced Vector Extensions
BIOS	Basic Input/Output System
CC	Common Criteria
CPU	Central Processing Unit
DMA	Direct Memory Access
EAL	Evaluation Assurance Level
EPT	Intel® Extended Page Tables
I/O MMU (IOMMU)	Input/Output Memory Management Unit
IPC	Inter Process Communication
IRQ	Interrupt Request
ITAS	In-Task Service
L4Re	L4 Runtime-Environment
L4Re SSK	L4Re Secure Separation Kernel CC 1.0.2
MMU	Memory Management Unit
MMIO	Memory-mapped I/O
NVMe	Non-Volatile Memory (NVM) Express
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PIO	Programmed Input/Output
PP	Protection Profile
RAM	Random Access Memory
ROM	Read-Only Memory
RTC	Real-Time Clock
SAR	Security Assurance Requirement
SDK	Software Development Kit
SFP	Security Function Policy

SFR	Security Functional Requirement
SMM	System Management Mode
SMMU	ARM System MMU
SR-IOV	Single Root Input/Output Virtualization
SSP	System Security Policy
SSSE	Supplement Streaming SIMD Extension
ST	Security Target
TOE	Target of Evaluation
TLB	Translation Lookaside Buffer
TSF	TOE Security Function
TSFI	TSF Interface
TSP	TOE Security Policy
UTCB	User Thread Control Block
VM	Virtual machine
VT-x	Intel® Virtualization Technology for x86 processors

10 References

CC: Common Criteria for Information Technology Security Evaluation, 3.1 Revision 5, April 2017

