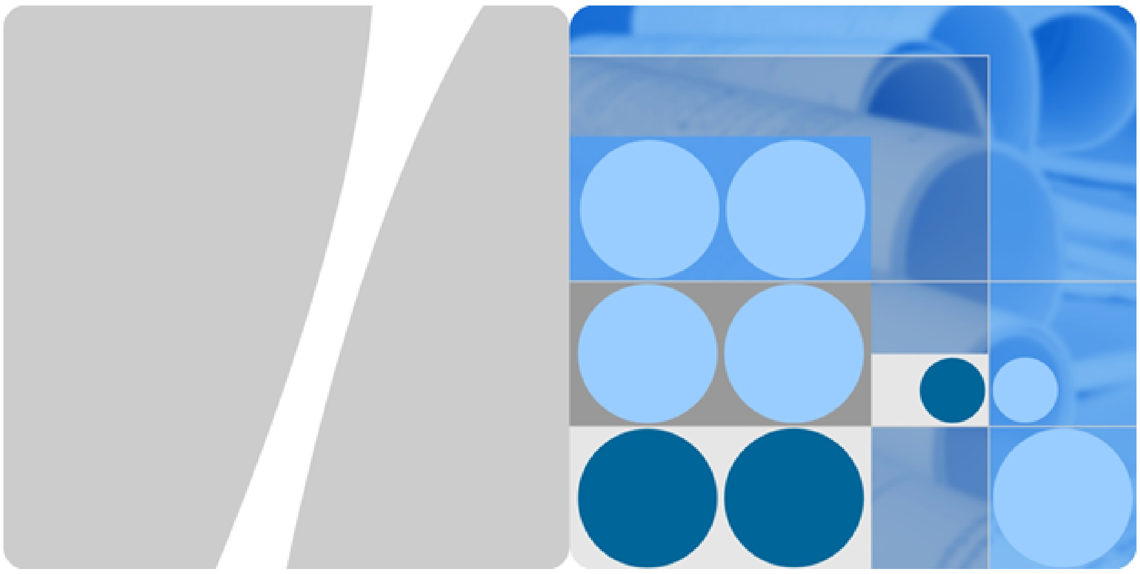

Huawei HongMeng Common Criteria Evaluation

ST: Security Target for Specified Hardware

Version	2.8
Status	Released
Last update	2019-07-23
Classification	Public

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.huawei.com

Change History

Date	Version	Section Number	Change Description	Author
2018.12.25	1.0	All	Initial Draft	Huawei OS Kernel Lab
2019.01.25	1.1	1, 4, 5, 6, 7	Add Contents of Platform Attestation	Huawei OS Kernel Lab
2019.02.07	1.2	All	Address Action Items	Huawei OS Kernel Lab
2019.02.22	2.0	All	Confirm Action Item List	Huawei OS Kernel Lab
2019.02.26	2.1	3, 4	Fix A.ENVIRONMENT, OE.ENVIRONMENT and OSP	Huawei OS Kernel Lab
2019.03.02	2.2	6	Update Some SFRs	Huawei OS Kernel Lab
2019.03.08	2.3	6, 7	Fix Inconsistencies between Implementation and SFRs	Huawei OS Kernel Lab
2019.03.12	2.4	7	Fix Action Items: A.ASE.28, A.ASE.29 and A.ASE.30	Huawei OS Kernel Lab
2019.03.13	2.5	4.3	Fix Inconsistency in Table 3	Huawei OS Kernel Lab
2019.04.04	2.6	7.2	Explain Enforcement of Integrity of Kernel Objects	Huawei OS Kernel Lab
2019.04.07	2.7	7.2	Fix Inconsistency in Section 7.2.1	Huawei OS Kernel Lab
2019.07.23	2.8	1, 10	Update Versions of TOE and References	Huawei OS Kernel Lab

Contents

List of Figures

List of Tables

1 Security Target Introduction

1.1 Security Target Reference

Title:	Huawei HongMeng Common Criteria Evaluation ST: Security Target for Specified Hardware
Author:	Huawei OS Kernel Lab
CC Version:	3.1 (Revision 5)
Assurance Level:	EAL 5+
Version:	2.8
Reference:	HongMeng Security Target
ITSEF:	Brightsight
Certification Body:	Netherlands Scheme for Certification in the Area of IT Security (NSCIB)
Keywords:	Operating System, Micro-kernel

1.2 TOE Reference

Name:	HongMeng
Developers:	Huawei OS Kernel Lab
Version:	1.2
Type:	Micro-kernel
User Guidance:	Huawei HongMeng Common Criteria Evaluation AGD_OPE: Operational User Guidance [OPE]
	Huawei HongMeng Common Criteria Evaluation AGD_PRE: Preparative Procedures [PRE]
Hardware Platform:	Huawei Kirin 970/980 SoC

1.3 TOE Overview

1.3.1 TOE Usage and Major Security Features

HongMeng[1], as our target of evaluation (TOE), is a micro-kernel that provides fine-grained resource management for applications running on top of it. More precisely, HongMeng is designed and implemented in such a way that it provides the means to hold:

- Confidentiality property: the resources of a subject (code, data, registers, interrupts, devices) cannot be observed by other subjects without an explicit authorization.
- Availability property: the system resources (code, data, registers, interrupts, devices) can be used by a subject (i.e. process) if and only if the subject is allowed by the TOE based on a priority-based queue.

By providing confidentiality and availability, HongMeng enforces secure access control and isolation for system resources. Thus it is a desired micro kernel for security scenarios.

In this document, subjects are mainly processes and a process is an instance of an application running on top of HongMeng. A process starts with some system resources, including but not limited to the initial code, data, priority and thread. Once it is executed, it may create a few threads running concurrently and also it acquires or releases some resources during its lifetime as well.

[1]In the following, the TOE and HongMeng are used interchangeably.

HongMeng is supposed to run on a real mobile device (e.g., mobile phone) with hardware support for ARM TrustZone. Since TEE is not the only zone on mobile devices, it is common to see multiple unconstrained applications running on the non-TEE side. However even in that case, those unconstrained applications cannot interfere with TOE, neither the applications running on top of TOE.

To achieve the security targets we described above, HongMeng employs the capability-based security model to control usage of the particular system resources and/or services (e.g., the memory, processor, communication channels, etc.). A capability is a transferable, unforgeable token that represents authority. It refers to a value that uniquely references an object along with an associated set of access rights. By virtue of its possession by a process that uses the referenced object, the capability token grants that process the capability to interact with an object in certain ways.

For example, HongMeng enforces restrictions on communications between separated processes by delegating the capability of communication channels with thoughtful rights setup.

The TOE comprises of:

- Software used to provide the HongMeng security functionalities.
- The guidance for the secure usage of the TOE after delivery.

1.3.2 TOE Type

HongMeng is a micro-kernel that provides fine-grained resource management for processes. HongMeng includes non-trivial mechanisms like Inter Process Communication (IPC), memory allocations, locking, device accessing, etc. A full-fledged operating system is implemented using such mechanisms. In this version, HongMeng does not include a file system or a device manager. Only very critical drivers are included in HongMeng.

In a real mobile scenario, system components (e.g., the file system, device manager, device drivers, etc.) are implemented as applications running on top of HongMeng. These additional components, together with HongMeng constitute a full-featured operating system to support TEE.

1.3.3 Required Non-TOE Hardware/Software/Firmware

The TOE can run on Huawei Mate/P series mobile phones with a minimum set of the following non-TOE hardware and software in the mobile phone.

Required non-TOE hardware:

- Huawei Kirin 970/980 SoC.
- RAM (6GB or more), which can be configured with a secure region to be accessed only by trust zone.
- NVM (64GB or more), which contains sensitive information for secure boot, Android OS and HongMeng image.

Required non-TOE software:

- Bootloader: It loads the kernel image into the memory and transfers the control to the kernel.
- ARM Trusted Firmware: It starts the trusted software (including the bootloader and the kernel) in the trusted zone and is responsible for the switch between the secure world and the non-secure world.
- System applications: Essential system applications for providing POSIX interfaces and interaction functionalities with the non-secure side.

Also, as we already mentioned in the paragraph of TOE Type, the TOE does not include any hardware or user applications.

1.4 TOE Description

1.4.1 Architecture

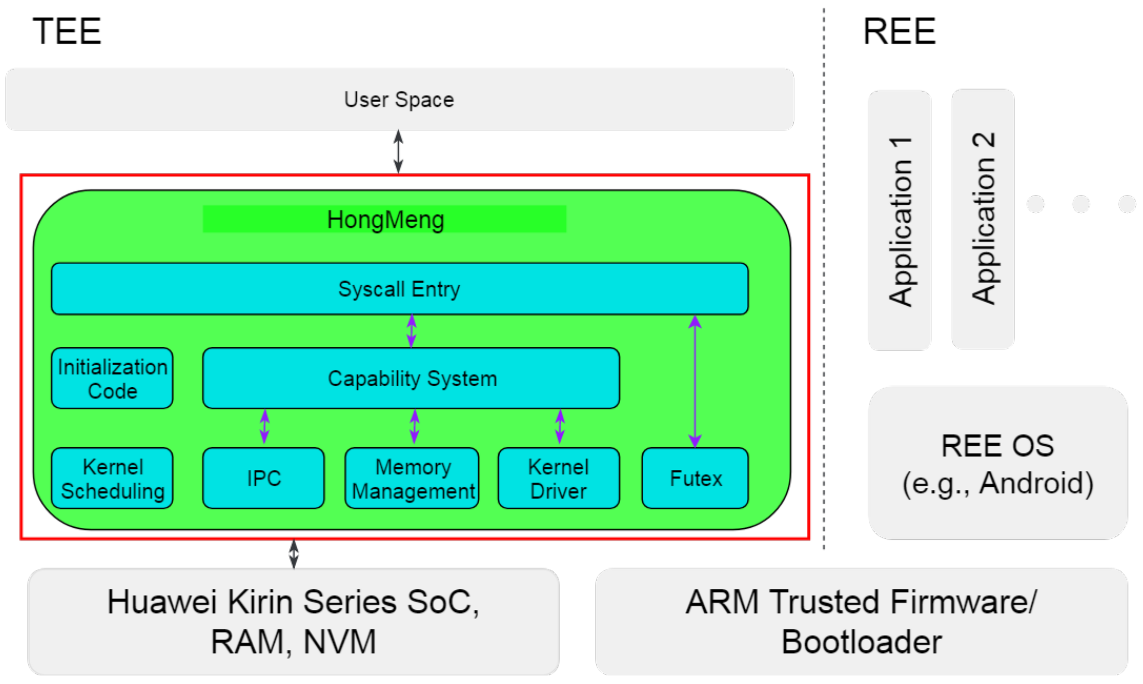

Figure 1: TOE and Operational Environment

As shown in Figure 1, the TOE consists of the following components:

- Initialization Code
- Inter Process Communication (IPC)
- Memory Management
  - Virtual Memory Management (Address Space)
  - Physical Memory Management
- Thread Management
  - Kernel Scheduling
  - Kernel Support for Synchronization (Futex)
- Kernel Driver
- Capability System

1.4.1.1 Initialization Code

The initialization code is executed at the very beginning of the TOE. First, the devices and resources are initialized according to the initial vectors and the device tree. Afterwards, it initializes other modules. Finally, it creates the first process and transfers the control to it.

1.4.1.2 Inter Process Communication

The TOE provides two different methods for processes to communicate with each other, named EBBCall and Async IPC.

EBBCall EBBCall is designed to provide an event-based synchronous call-receive-reply IPC paradigm. In EBBCall, a group of threads that intend to receive messages, which are identified as servers, can be organized as a ServerPool. To send a message to the servers, a client thread can initiate a Call operation on a ServerPool with a piece of message. Based on a preset policy, the TOE will choose an available server within this ServerPool to serve this message, and a pre-defined handler will be invoked based on the message type. While executing the handler, the client will stay blocked. During or after the handling procedure, the server can invoke a Reply operation to send a reply message back to the client, and the client thread will be unblocked with the reply message payload.

Async IPC Async IPC allows one thread to asynchronously send messages without waiting for the other end to receive. Senders and receivers are organized via Channels. A receiver can invoke Receive on a Channel to wait for a message. Upon receiving a message, the receiver will be unblocked with the message payload.

1.4.1.3 Memory Management

Memory management includes management of physical memory and management of virtual memory address spaces.

In physical memory management, the TOE assigns all available physical memory regions to user processes (called owners) after initialization. A physical memory region's owner is able to decide what the physical memory region is used for. It can map the physical memory region to store user data or tell the TOE to use this physical memory region to store specified kernel objects. For security reasons, the TOE records the information of all physical memory regions and always guarantees the physical memory isolation between user space and kernel. The TOE also allows user processes to split physical memory regions and change ownership to support memory management in user space.

As for the management of virtual memory address space, the TOE only manages page tables. In the TOE, each virtual memory address space is statically divided into the kernel part and the user part. User processes can arbitrarily map (unmap) specified virtual memory regions in the user part to (from) specified physical memory regions through syscalls. The TOE validates the virtual memory regions and physical memory regions so that user processes cannot operate on virtual memory address spaces without permissions or touch illegal physical memory regions.

1.4.1.4 Thread Management

The TOE provides a combined set of mechanisms to manage threads.

Kernel Scheduling Kernel scheduling is a TOE module which decides the thread to run at a certain point of time. The TOE implements the priority-based FIFO (first in, first out) scheduling. Upon scheduling, the scheduler finds a ready thread with the highest priority and puts it to run on the current CPU. Each thread has a timeslice to keep track of the current remaining time on the CPU, which will be decreased by 1 upon each timer tick. When the timeslice is used up, the current thread will give up the CPU and be appended to the tail of scheduler queue. At this point the scheduler will choose a new thread run. Note that in HongMeng, the current running thread also gives up the CPU whenever a syscall occurs or preempted by a thread with higher priority.

Kernel Support for Synchronization HongMeng provides a light-weight method for process synchronization, i.e., Fast Userspace muTEXes (Futex). Futex can be used to implement the basic locking, or as the base for building more advanced locking mechanism, such as the semaphores, mutex locks or condition variables.

A Futex consists of a kernel space wait queue that is attached to an atomic integer in user space. Multiple threads operate on the integer entirely in user space, and only resort to relatively expensive system calls to request operations on the wait queue (for example, to wake up waiting processes when the lock is free, or to put the current process on the wait queue when the lock is contended). So the process will not be trapped in the kernel space if there is no contention, and this improves the kernel efficiency.

1.4.1.5 Kernel Driver

Only very critical drivers are included in HongMeng, while most of the drivers are implemented in the user space.

HongMeng now contains the following kernel drivers:

- GIC driver: It provides the abstraction of the GIC interrupt controller:
  - It gets the version of the GIC device from device tree, and initializes it accordingly.
  - It provides the interfaces to operate on the GIC device: such as masking/unmasking the specific interrupts, setting the priorities of the interrupts, reading the interrupt number, etc.

  The kernel reads the interrupt request number from the GIC driver and then delivers it to the appropriate handler to process it.
- Timer driver: It initializes the hardware timer on the platform which periodically triggers timer interrupt. The timer interrupt is used for scheduling in TOE.

1.4.1.6 Capability System

The capability system implements the capability-based security mechanism to provide the basic access control for the resources. It protects the resources owned by one process from being modified or tampered with by other processes. The resources are represented as the kernel objects in HongMeng. To operate on the kernel object, one process should be authorized with the corresponding permission first. In addition to isolation, the capability system also allows to share the resources among processes by permitting one process to grant the authorization to other processes with restricted rights.

Capability Rules A capability is an unforgeable token that points to a specific kernel object, and carries access rights that control which methods can be called on object. Capabilities exist in the space of the corresponding process, and the process can use a reference pointing to a certain capability to request the kernel object. In general terms, the operating rules of capability are listed as follows:

- All kernel objects have corresponding capabilities.
- To perform an operation on an object, a process must hold a capability in its possession that has sufficient access rights for the requested services.
- The kernel shall not reject any capability service invocation from a legitimate owner of this capability, and guarantees that the requested service will be fulfilled.

Capability of Process The storage of capabilities of a process is separated from other processes, and other processes cannot modify or tamper with it without an explicit authorization.

The initial capabilities are given to a process when it is created. Afterwards, only the process can create capabilities for itself. When a capability is created, a corresponding kernel object is created at the same time. A process can give its own capability to other processes and it can revoke the authorization at any time.

Support for Other Modules Capability mechanism gives the support for the security of other modules:

- Memory management: A page table is a kernel object owned by the individual process and only the owner has the authorization at the beginning. It cannot be modified without awareness of the owner, so the address space cannot be damaged by a malicious process.

- IPC: The channel used for communication is a kernel object. Only the processes which have the capability to read from (or write to) the channel can receive messages from (or send to) the channel. So the contents of the messages cannot be accessed by a malicious process.

1.4.2 TOE Operational Environment

Hardware is the physical part of operational environment of the TOE. In general, hardware includes CPUs, RAM and NVM. The physical part also comprises the boot loader that is used to load the TOE on the hardware, and transfers its control authority from the TOE afterwards.

In a mobile operating system (e.g., TEE), the TOE may share the hardware resource with REE OS (e.g., Android). In such case, the TOE is an isolated system that runs in parallel with the user-facing operating system. Trusted Applications (TAs) based on the TOE are isolated to the ones in REE OS, while the communication may still be possible.

1.4.3 TOE Physical Boundary

The TOE is a software product. In Figure 1, each component within the red block is within the TOE physical boundary, and each component outside the red block is outside of the TOE physical boundary. Therefore, hardware and firmware/bootloader do not belong to the TOE.

The TOE also includes guidance documents. Table 1 shows the physical scope of the TOE.

Type	Delivery Item	Version
Software	Internal Product Name: OS Kernel TD External Product Name: HongMeng Format: hm_release.tar.gz Info: User can login the Version Management Platform (VMP) to download the software binary archive in accordance to the version of the TOE. User can verify the software by digital signature. The digital signature is also published on the VMP website.	Internal Version Number: V100R001C00SPC900B006 External Version Number: v1.2
Product Guidance	Huawei HongMeng Common Criteria Evaluation AGD_OPE: Operational User Guidance [OPE] Format: HongMeng_AGD_OPE-v2.0.pdf Info: The document is delivered on VMP website in accordance to the version of the TOE.	2.0
	Huawei HongMeng Common Criteria Evaluation AGD_PRE: Preparative Procedures [PRE] Format: HongMeng_AGD_PRE-v1.4.pdf Info: The document is delivered on VMP website in accordance to the version of the TOE.	1.4

Table 1: Physical Scope

1.4.4 TOE Logical Boundary

The TOE provides following security services.

- User identification and TSF protection

  All processes have to be identified before performing any TSF-mediated activity. Meanwhile, the TOE preserves a secure state when the TOE panic occurs.

- Capability-based access control

  In the TOE, each kernel object is associated with a *capability*. To access a kernel object, a process should show the *capability* associated with the specific rights of the kernel object explicitly.

- Information flow control and residual information removal

  Only authorized processes can exchange information via IPC. In addition, any residual information in message related structures will be filled with zero before use.

- Memory management

  The TOE exploits memory management to achieve fine-grained access control for physical memory and virtual memory.

- Thread management

  By means of kenrel scheduling and Futex functionalities, that ensure that only processes with a highest priority are allowed to use the resources.

- Safe and secure state preservation

  The TSF supports reaching and keeping a safe and secure state of the TOE. After the TOE has been successfully initialized, the global configuration is correct and matches the hardware conditions.

  The TSF ensures also that the TOE is not compromised in the event of a failure during operation by means of logical checks on key paths of the code.

- Platform attestation

  The TOE provides the means to uniquely identify the underlying platforms.

1.4.5 Reference Device Life Cycle

The device life cycle outlined here is an overall life cycle from which implementations cannot deviate according to development, manufacturing and assembly processes. The life cycle is split into following phases:

- Phase 1 corresponds to the design of software TOE.
- Phase 2 corresponds to the overall design of the software, firmware and hardware platform supporting the TOE.
- Phase 3 covers software preparation (linking the TOE and other software in user space).
- Phase 4 consists of device assembling. It includes any initialization and configuration step necessary to bring the device to a secure state prior to delivery to the end-user.
- Phase 5 stands for the end-usage of the device.

Table 2 shows the whole life cycle of the TOE device.

The phase related to the TOE during the whole device life cycle is as below:

Phase 1 Software design: Including the TOE software design and development, and delivering source code of the TOE to system integrator.

The other phases are out of scope of this ST.

Phase	Actors
1 & 2: Firmware/ Software/Hardware Design	The TOE developer: 1. is in charge of the TOE software development and testing. 2. does not develop the trusted firmware that instantiates/initializes the the TOE (e.g. part of the secure boot code). It is developed by hardware/firmware team and not included in this document. 3. specifies the TOE software linking requirements. The TEE software designer (not the TOE developer) is in charge of TEE software development and testing compliant with GlobalPlatform specification, which is not the scope of the TOE developer's responsibilities. The device manufacturer (not the TOE developer) may design additional REE software that will be linked with the TEE in Phase 4 to provide REE controlled resources. Both the device manufacturer (not the TOE developer) and TEE software designer (not the TOE developer) may design Trusted Applications that they will integrate in Phase 4. The TEE hardware designer is in charge of designing (part of) the processor(s) where the TEE software runs on and designing (part of) the hardware security resources used by the TEE. The silicon vendor designs the NVM and the secure portion of the TEE chipset. If the silicon vendor is not designing the full TEE hardware, the silicon vendor integrates (and potentially augments) the TEE hardware designed by the TEE hardware designer(s).
3: System Integrator	The system integrator is responsible for the integration, validation and preparation of the software to load in the product that will include the TOE, any pre-installed application, and additional software required to use the product (e.g. REE, Client Applications).
4: Device Manufacturing	The device manufacturer is responsible for the device assembling and initialization, and any other operation on the device (including loading or installation of applications) before delivery to the end user.
5: End-usage Phase	The end-user obtains a device ready for use.

Table 2: Actors in Device Life Cycle

2 Conformance Claims

2.1 CC Conformance Claim

This Security Target conforms to CC Part 2 extended and Part 3 conformant, with a claimed Evaluation Assurance Level of EAL 5, augmented by ALC_FLR.1.

This Security Target claims conformance to the following specifications:

- Common Criteria for Information Technology Security Evaluation Part 2: Security Functional Requirements, Version 3.1, Revision 5, April 2017.
- Common Criteria for Information Technology Security Evaluation Part 3: Security Assurance Requirements, Version 3.1, Revision 5, April 2017; augmented by ALC_FLR.1.

2.2 PP Claim

This Security Target does not claim conformance to any PP.

3 Security Problem Definition

HongMeng is designed to enforce access control policy for all subjects (processes) running on top of it and to enforce that all subjects use the available resources under control of HongMeng.

Thus security problems here can be described, based on different type of assets, by how a certain kind of information might leak which breaks confidentiality, and how the processing resources is monopolized by a process.

In this section, we first categorise the assets from subject's (process's) perspective and then address all the potential ways a malformed application might threaten confidentiality and availability for each particular asset.

3.1 Assets

Assets are categorised as **user data**, **TSF data** and **processing resources**. User data comprises AS.PROC.DATA, AS.PROC.REG and AS.PROC.CODE. TSF data comprises AS.TSF.DATA and AS.TSF.CODE.

AS.PROC.DATA

The private data of an individual process, including the heap, the stack and the shared memory.

AS.PROC.REG

The registers of a process are stored in CPU while the process is running.

AS.PROC.CODE

The code of a process.

AS.TSF.DATA

The data managed by the TSF, including the kernel objects used by kernel modules, such as IPC, scheduler, kernel driver, Futex, etc.

AS.TSF.CODE

The code of TSF, including the code performing the system call and enforcing the security policy.

The underlying platform is protected from the user space perspective by means of the TSF and is grouped in the asset defined below:

AS.RESOURCE.PHYSICAL

The processing resources of the platform managed by the kernel by means of the TSF with AS.TSF.DATA (priority).

3.2 Users/Subjects

S.PROC

The users/subjects are the processes in the user space.

3.3 Threats

This section of the security problem definition shows the threats that are to be countered by the TOE, its operational environment, or a combination of the two.

We assume that an attacker is a threat agent (a person or a process acting on his/her behalf) trying to undermine the TOE security policy defined by the current ST and, hence, the TSF.

We also assume that an attacker will only carry out software attacks and attacks from other sources than software shall be averted by the TOE operational environment or by an organisational security policy.

T.UNAUTHORIZED_ACCESS

An attacker is able to read or modify user data and/or TSF data whose access is not authorized to the subject.

T.QUEUE_SKIPPING

An attacker attempts to continuously execute threads using all the processing resources bypassing the scheduler, thus preventing other threads in the queue to be executed.

3.4 Organizational Security Policies

This section presents the organizational security policies that have to be implemented by the TOE and/or its operational environment.

OSP.SYSTEM_INTEGRATOR

1. The system integrator shall verify hardware from Huawei Kirin 970/980 SoC.
2. The system integrator shall ensure that the environment (either software, hardware, the OSP for the product in field, or by a mix of these) ensures following properties:
   (a) The bootloader shall initialize the hardware so the TOE starts in a safe and secure state.
   (b) The integrity of the product binary image is ensured when the TOE is loaded and starts running.
3. The system integrator shall correctly perform the integration process according to the TOE guidance [PRE].
4. The system integrator shall install the integrated product binary image on the hardware.

OSP.SAFE_SECURE_STATE

The TOE shall reach a safe and secure state after the bootloader has successfully verified the integrity of the TOE, initialized the hardware required by the TOE, performed the hand-over to the TOE and after the TOE is successfully initialized.

3.5 Assumptions

This section states the assumptions that hold on the TOE operational environment. These assumptions have to be satisfied by the operational environment.

A.TRUSTWORTHY_PERSONNEL

The personnel configuring, integrating and installing the TOE (system integrator) are trustworthy. In particular, the system integrator should ensure following properties:

- The system integrator shall correctly perform the integration process according to the TOE guidance [PRE]. operational policy (and, if necessary, according to the hardware manuals).
- The system integrator shall install the integrated product binary image on the hardware.

A.ENVIRONMENT

The environment (either software, hardware, or by a mix of both) should ensure following properties:

- The bootloader shall initialize the hardware so the TOE starts in a safe and secure state.
- The memory managed by the TOE, and the Kirin 970/980 SoC required by the TOE shall be protected in confidentiality and integrity from the outside of the TOE.

- The bootloader and ARM Trusted Firmware shall be protected in integrity from the outside of the TOE.

A.TRUSTED_PROCESS

It is assumed that only trusted processes will be allowed to directly use TOE services and data. Specifically, only those processes bound to the TOE before its operational use are assumed to be trusted.

Note: the trusted processes will be those which conform the whole OS as part of the kernel components in the user space, and which will be configured during the integration phase before the TOE is in the final user phase.

4 Security Objectives

4.1 Security Objectives for the TOE

This section states the security objectives for the TOE. The following objectives must be satisfied by the TOE:

O.CONFIDENTIALITY

The TOE shall preserve confidentiality of assets with a confidentiality need.

O.ACCESS_CONTROL

The TOE shall grant read and/or write permissions to assets only to authorised users. Specifically, only the owner and processes granted to read or modify data , and only those processes which establish an IPC buffer to exchange messages shall be allowed access user data and/or TSF data.

O.SAFE_SECURE_STATE

The TOE shall preserve a secure state when a failure occurs.

O.PRIORITY

The TOE shall define a priority based scheduling for the threads. In particular, only scheduled threads in the queue with the highest priority can use resources of the system.

O.ATTESTATION

The TOE shall uniquely identify the underlying platforms. In particular, the underlying platforms are Huawei Kirin 970/980 SoC.

4.2 Security Objectives for the Operational Environment

This section states the security objectives for the TOE operational environment covering all the assumptions and the organizational security policies that apply to the environment. The following security objectives apply to any TOE operational environment that does not implement any additional security feature.

OE.TRUSTWORTHY_PERSONNEL

The personnel (especially, the system integrator) configuring, integrating and installing the TOE are trustworthy.

OE.ENVIRONMENT

The environment ensures the following properties:

- The bootloader initializes the hardware so the TOE starts in a safe and secure state.
- The memory managed by the TOE, and the Kirin 970/980 SoC required by the TOE are protected in confidentiality and integrity from the outside of the TOE.
- The bootloader and ARM Trusted Firmware are protected in integrity from the outside of the TOE.

OE.TRUSTED_PROCESS

The key processes that are running on the TOE shall be providing necessary user space services correctly and ensure their own integrity while running.

4.3 Security Objectives Rationale

Table 3 provides mappings between TOE Security Problem Definition (SPD) and Security Objectives.

	O.CONFIDENTIALITY	O.ACCESS_CONTROL	O.PRIORITY	O.SAFE_SECURE_STATE	O.ATTESTATION	OE.TRUSTWORTHY_PERSONNEL	OE.ENVIRONMENT	OE.TRUSTED_PROCESS
T.UNAUTHORIZED_ACCESS	√	√		√		√	√	
T.QUEUE_SKIPPING			√	√		√	√	
OSP.SYSTEM_INTEGRATOR					√	√	√	
OSP.SAFE_SECURE_STATE				√				
A.TRUSTWORTHY_PERSONNEL						√		
A.ENVIRONMENT							√	
A.TRUSTED_PROCESS								√

Table 3: Security Objectives Rationale

4.3.1 Security Objectives Rationale: Threats

4.3.1.1 Threat: T.UNAUTHORIZED_ACCESS

O.CONFIDENTIALITY prevents the disclosure of user data during normal operation of the TOE. O.ACCESS_CONTROL prevents the unauthorized access to user data and/or TSF data during normal operation of the TOE, and O.SAFE_SECURE_STATE prevents the unauthorized access to data when the TOE is in the failure state.

In addition, OE.TRUSTWORTHY_PERSONNEL and OE.ENVIRONMENT ensure that the TOE is configured and used in a secure state.

4.3.1.2 Threat: T.QUEUE_SKIPPING

O.PRIORITY ensures that all processes using the TOE are executed based on a priority based queue, and O.SAFE_SECURE_STATE prevents the execution of processes out of the queue when the TOE is in the failure state.

In addition, OE.TRUSTWORTHY_PERSONNEL and OE.ENVIRONMENT ensure that the TOE is configured and used in a secure state.

4.3.2 Security Objectives Rationale: Organizational Security Policies (OSPs)

4.3.2.1 OSP: OSP.SYSTEM_INTEGRATOR

The objective OE.TRUSTWORTHY_PERSONNEL and OE.ENVIRONMENT directly enforce this OSP. The objective O.ATTESTATION supports the enforcement of this OSP by providing a means to identify the underlying platform.

4.3.2.2 OSP: OSP.SAFE_SECURE_STATE

The objective O.SAFE_SECURE_STATE directly enforces this OSP.

4.3.3 Security Objectives Rationale: Assumptions

4.3.3.1 Assumption: A.TRUSTWORTHY_PERSONNEL

The assumption A.TRUSTWORTHY_PERSONNEL is upheld by OE.TRUSTWORTHY_PERSONNEL directly.

4.3.3.2 Assumption: A.ENVIRONMENT

The assumption A.ENVIRONMENT is upheld by OE.ENVIRONMENT.

4.3.3.3 Assumption: A.TRUSTED_PROCESS

The assumption A.TRUSTED_PROCESS is upheld by OE.TRUSTED_PROCESS.

5 Extended Components Definition

5.1 Definition of Family FAU_SAS

To define the security functional requirements of the TOE an additional family (FAU_SAS) of the Class FAU (Security Audit) is defined here. This family describes the functional requirements for the storage of audit data. It has a more general approach than FAU_GEN, because it does not necessarily require the data to be generated by the TOE itself and because it does not give specific details of the content of the audit records.

The family "Audit data storage (FAU_SAS)" is specified as follows.

5.1.1 FAU_SAS Audit data storage

Family behaviour

This family defines functional requirements for the storage of audit data.

Component levelling

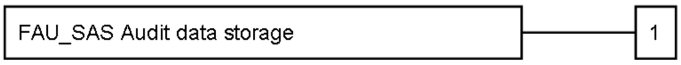

Figure 2: FAU_SAS

FAU_SAS.1 Requires the TOE to provide the possibility to store audit data.

Management: FAU_SAS.1

There are no management activities foreseen.

Audit: FAU_SAS.1

There are no actions defined to be auditable.

5.1.1.1 FAU_SAS.1 Audit Storage

Hierarchical to:	No other components.
Dependencies:	No other components.

FAU_SAS.1.1 The TSF shall provide the capability to identify [assignment: *list of subjects*] and store [assignment: *list of audit information*] in the [assignment: *type of kernel object*].

6 Security Requirements

6.1 Introduction

6.1.1 Conventions

The CC allows following operations to be performed on security requirements on the security component level [CC2]:

- Assignment: The assignment operation is used to set the unspecified parameter with a specified value, for example, the list of actions. The assignments having been made by the ST author are *italicized text*.
- Selection: The selection operation is used to select one or more items in a given component. The selections having been made by the ST author are <u>underlined text</u>.
- Refinement: The refinement operation is used to present more details in security requirement, in order to restrict the requirement or satisfy editorial reasons. The refinements having been made by the ST author are **bold text**.
- Iteration: The iteration operation is used to describe multiple requirements based on the same component. The iterations are identified by a suffix with slash added to the security requirement.

Other conventions used in descriptions of SFRs are as follows:

- Application Notes: Notes added by the ST author are called "Application Note" which are enumerated as "a", "b", . . . and are formatted with underline such as "<u>Application Note a:</u> AppNote Description".
- References: Indicated with [square brackets].

6.1.2 Definitions

The security functional components [CC2] used in this Security Target are shown in the following table.

Security Functional Components	Component Identification
FIA_ATD.1	User attribute definition
FIA_UID.2	User identification before any action
FIA_USB.1	User-subject binding
FMT_SMR.1	Security roles
FDP_ACC.1/CAP	Subset access control (for Capability)
FDP_ACC.1/MEM	Subset access control (for Memory)
FDP_ACF.1/CAP	Security attribute based access control (for Capability)
FDP_ACF.1/MEM	Security attribute based access control (for Memory)
FDP_IFC.1/EBB	Complete information flow control (for EBBCall)
FDP_IFC.1/ASY	Complete information flow control (for Async IPC)
FDP_IFF.1/EBB	Simple security attributes (for EBBCall)
FDP_IFF.1/ASY	Simple security attributes (for Async IPC)
FDP_RIP.1	Subset residual information protection
FMT_MSA.1/EBB	Management of security attributes (for EBBCall)
FMT_MSA.1/ASY	Management of security attributes (for Async IPC)
FMT_MSA.1/CAP	Management of security attributes (for Capability)

Security Functional Components	Component Identification
FMT_MSA.1/MEM	Management of security attributes (for Memory)
FMT_MSA.3/EBB	Static attribute initialization (for EBBCall)
FMT_MSA.3/ASY	Static attribute initialization (for Async IPC)
FMT_MSA.3/CAP	Static attribute initialization (for Capability)
FMT_MSA.3/MEM	Static attribute initialization (for Memory)
FMT_SMF.1/EBB	Management functions (for EBBCall)
FMT_SMF.1/ASY	Management functions (for Async IPC)
FMT_SMF.1/CAP	Management functions (for Capability)
FMT_SMF.1/MEM	Management functions (for Memory)
FPT_FLS.1	Failure with preservation of secure state
FRU_PRS.1	Limited prioritized CPU time
FAU_SAS.1	Audit storage

Table 4: Security Functional Components

The statement of the security functional requirements relies on the following characterization of the TOE in terms of users, subjects, objects, information, user data, TSF data, operations and their security attributes (cf. CC Part 1 [CC1] for the definition of these notions).

Users and subjects stand for entities outside the TOE, which refers to any process in the user space, denoted as S.PROC.

Objects stand for kernel objects inside the TOE:

- *OB.CNODE*: The collection of the *capabilities* that a process owns, where each *capability* is associated with a kernel object.
- *OB.THREAD*: Information set of a thread, such as priority, affinity, state, etc.
- *OB.CHANNEL*: The message channel for asynchronous communication containing the message itself and information about the sender etc. A process may send a message to a specific channel, and another process may receive the message from the particular channel.
- *OB.SERVERPOOL*: Kernel object consisting of a set of threads which may receive and process the specific synchronous messages, and information about the incoming synchronous messages.
- *OB.REQUEST*: The message for the synchronous communication and interrupt delivery.
- *OB.VSPACE*: Address space of a user process.
- *OB.PMEM*: The attributes of a segment of physical memory, for example, the base physical address, memory size, etc.
- *OB.SYSCTRL*: Kernel object containing system information to provide system control functions to the user.
- *OB.IRQCTRL*: Kernel object containing interrupt configuration to provide system interrupt control functions to the user.

Information stands for data exchanged between subjects:

- *I.MSG*: the message sent by the subjects through inter process communication mechanism.

TSF data consists of runtime TSF data and TSF persistent data necessary to provide the security services. It includes all the security attributes of users, subjects, objects and information.

The security attributes are defined as follows:

- *ATTR.PROC.ID* ::= PID: The PID of the process.
- *ATTR.OB.OWNER* ::= PID: Represents the process which the kernel object belongs to.
- *ATTR.OB.SVISITOR* ::= {PID}: Represents a set of processes which have the supervised access to the kernel object.
- *ATTR.OB.PERM*: Maps from processes to the rights. The rights represent the functions that a process can invoke.
- *ATTR.PMEM.STATUS* ::= one of {MAP_ONLY, FREE, MAPPED, KERNEL, VOID}: Represents the status of a physical memory region.

  MAP_ONLY means user processes can only do mapping on it, e.g. used for MMIO, shared with external system.

  FREE means it is not being used.

  MAPPED means it is being mapped by user processes.

  KERNEL means it is being used to store kernel data.

  VOID means there is no physical memory located in this region.

The authorized security roles for each kernel object are defined as follows:

- *R.OWNER*: The owner of a kernel object, the one who creates it. It can grant the corresponding kernel object with the specific permissions to other processes, or delete it.
- *R.VISITOR*: The visitor of a kernel object. The ability to access that kernel object is granted from the owner. It cannot modify the security attributes or delete the kernel object. The authorization can be revoked at anytime by the owner.
- *R.SVISITOR*: The supervised visitor of a kernel object. The ability to access that kernel object is gained by possessing the supervised access right to the *OB.CNODE* of that kernel object. It has the same privilege of *R.OWNER* of that kernel object, includes granting the kernel object to a *R.VISITOR*, but does not own it.

The operations on kernel objects are defined as follows:

- *OP.GRANT*: The owner and the supervised visitor of a kernel object grants the ability to access that kernel object to the visitor.
- *OP.REVOKE*: The owner and the supervised visitor of a kernel object cancels the authorization of the kernel object from the visitor.
- *OP.REJECT*: The visitor of a kernel object gives up the authorization to access that kernel object.
- *OP.NEWCAP*: A new capability with a specified right is created with the associated kernel object.

The operations of the IPC are defined below:

- EBBCall:
  - *OP.CALL*: A message is delivered from one subject to another if the caller subject has access to an *OB.SERVERPOOL*.
  - *OP.REPLY*: A message is delivered from one subject to another if the replier has access to an *OB.REQUEST*.
  - *OP.SETCPUMASK*: A specific set of CPUs is set for *OB.SERVERPOOL* to process requests only for those CPUs.
  - *OP.SETPRIOMASK*: A specific set of priorities is set for *OB.SERVERPOOL* to process requests only for those priorities.

- Async IPC:
  - *OP.SEND*: A message is delivered to an *OB.CHANNEL*.
  - *OP.RECEIVE*: A message is grabbed from an *OB.CHANNEL*.

The operations about the memory are defined below:

- *OB.VSPACE*:
  - *OP.MAP*: The operation to establish the mapping from the specific virtual addresses to specific physical addresses of an *OB.PMEM* in an *OB.VSPACE*.
  - *OP.UNMAP*: The operation to erase a mapping in an *OB.VSPACE* on the specific virtual addresses.
- *OB.PMEM*:
  - *OP.REGISTER_TO_KERNEL*: The operation to specify an *OB.PMEM* to store kernel data, e.g., storing specified kernel objects.
  - *OP.FREE*: The operation to release an *OB.PMEM* from storing kernel data.
  - *OP.PLUGIN*: The operation to add a memory block for hot plugging.

The rights on operations are defined as follows:

- *P.SEND*: Right necessary for S.PROC to invoke *OP.SEND*.
- *P.RECEIVE*: Right necessary for S.PROC to invoke *OP.RECEIVE*.
- *P.MAP*: Right necessary for S.PROC to invoke *OP.MAP*.
- *P.UNMAP*: Right necessary for S.PROC to invoke *OP.UNMAP*.

6.1.3 Security Function Policies

This ST defines the following access control and information flow Security Function Policies (SFPs):

EBB IPC Information Flow Control SFP (*SFP.EBB*):

1. Purpose: To control the flow of the message from and to processes using *OB.SERVERPOOL*. This policy contributes to ensuring the confidentiality of communication messages (*I.MSG*).
2. Subjects: User processes.
3. Information: *I.MSG*.
4. Security attributes: The specific sender processes and the receiver processes.
5. SFR instances: FDP_IFC.1/EBB, FDP_IFF.1/EBB.

Asynchronous IPC Information Flow Control SFP (*SFP.ASY*):

1. Purpose: To control the flow of the message from and to processes using *OB.CHANNEL*. This policy contributes to ensuring the confidentiality of communication messages (*I.MSG*).
2. Subjects: User processes.
3. Information: *I.MSG*.
4. Security attributes: The specific sender processes and the receiver processes.
5. SFR instances: FDP_IFC.1/ASY, FDP_IFC.1/ASY.

Capability-based Access Control SFP (*SFP.CAP*):

1. Purpose: To control the access to kernel objects.
2. Subjects: User processes.
3. Objects: Kernel objects.
4. Security attributes: The capabilities corresponding to the kernel objects.
5. SFR instances: FDP_ACC.1/CAP, FDP_ACF.1/CAP.

Memory Access Control SFP (*SFP.MEM*):

1. Purpose: To control the access to virtual memory spaces and physical memory.
2. Subjects: User processes.
3. Objects: Memory resources through *OB.PMEM* and *OB.VSPACE*.
4. Security attributes: The access right to certain virtual memory range and physical memory.
5. SFR instances: FDP_ACC.1/MEM, FDP_ACF.1/MEM.

6.2 TOE Security Functional Requirements

This chapter provides the set of Security Functional Requirements (SFRs) the TOE has to enforce in order to fulfill the security objectives.

6.2.1 User Identification and TSF Protection

6.2.1.1 FIA_ATD.1 User attribute definition

Hierarchical to:	No other components.
Dependencies:	No dependencies.

FIA_ATD.1.1 The TSF shall maintain the following list of security attributes belonging to individual users: [assignment: *ATTR.PROC.ID, ATTR.OB.OWNER, ATTR.OB.SVISITOR, ATTR.OB.PERM*].

6.2.1.2 FIA_UID.2 User identification before any action

Hierarchical to:	FIA_UID.1 Timing of identification.
Dependencies:	No dependencies.

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

6.2.1.3 FIA_USB.1 User-subject binding

Hierarchical to:	No other components.
Dependencies:	FIA_ATD.1 User attribute definition.

FIA_USB.1.1 The TSF shall associate the following user security attributes with subjects acting on the behalf of that user: [assignment: *ATTR.PROC.ID, ATTR.OB.OWNER, ATTR.OB.SVISITOR, ATTR.OB.PERM*].

FIA_USB.1.2 The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: [assignment:

- *Each process is associated with a unique identifier ATTR.PROC.ID;*
- *Each process is associated with an initial OB.THREAD, an OB.CNODE and several OB.PMEMs. The process is the owner of these kernel objects and has entire permissions.*

].

FIA_USB.1.3 The TSF shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users: [assignment:

- *The process identifier ATTR.PROC.ID cannot be modified;*
- *The owner of the kernel object ATTR.OB.OWNER cannot be modified;*
- *Only the owner of the kernel object can modify the permission ATTR.OB.PERM.*

].

6.2.1.4 FPT_FLS.1 Failure with preservation of secure state

Hierarchical to:	No other components.
Dependencies:	No dependencies.

FPT_FLS.1.1 The TSF shall preserve a secure state when the following types of failures occur: [assignment: *Inconsistent state caused by transient failures during initialization and runtime*].

6.2.1.5 FMT_SMR.1 Security roles

Hierarchical to:	No other components.
Dependencies:	FIA_UID.1 Timing of identification.

FMT_SMR.1.1 The TSF shall maintain the roles [assignment: *R.OWNER, R.VISITOR, R.SVISITOR*].

FMT_SMR.1.2 The TSF shall be able to associate users with roles.

Application Note: The security roles maintained by the TSF are used for the following purposes associated to each SFP:

- *SFP.CAP*: Implementation of proper management of capability access.
- *SFP.EBB*: Limitation of the call functionality to the callers, and limitation of the serving functionality to the servers.

- *SFP.ASY*: Limitation of the notify functionality to the senders, and limitation of the receiving functionality to the receivers.
- *SFP.MEM*: Access limitation to given a physical memory region.

6.2.2 Capability-based Access Control

6.2.2.1 FDP_ACC.1/CAP Subset access control

Hierarchical to:	No other components.
Dependencies:	FDP_ACF.1/CAP Security attribute based access control.

FDP_ACC.1.1 The TSF shall enforce the [assignment: *SFP.CAP*] on [assignment:

- *Subjects: S.PROC*
- *Objects: kernel objects*
- *Operations: OP.NEWCAP, OP.GRANT, OP.REVOKE, OP.REJECT*

].

6.2.2.2 FDP_ACF.1/CAP Security attribute based access control

Hierarchical to:	No other components.
Dependencies:	FDP_ACC.1/CAP Subset access control
	FMT_MSA.3/CAP Static attribute initialisation.

FDP_ACF.1.1 The TSF shall enforce the [assignment: *SFP.CAP*] to objects based on the following: [assignment:

- *Subjects: S.PROC*
- *Objects: kernel objects*
- *Security attributes: ATTR.OB.OWNER, ATTR.OB.PERM, ATTR.OB.SVISITOR*

].

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [assignment:

- *The owner and the supervised visitor of the kernel object can OP.GRANT it to other processes.*
- *The owner and the supervised visitor of the kernel object can OP.REVOKE the authorization from the visitor.*
- *The visitor can OP.REJECT the authorization of the kernel object.*
- *A process can create a new kernel object (OP.NEWCAP).*

].

FDP_ACF.1.3	The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: [assignment: *None*].
FDP_ACF.1.4	The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [assignment: *None*].

6.2.2.3 FMT_MSA.1/CAP Management of security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/CAP Subset information flow control
	FMT_SMR.1 Security roles
	FMT_SMF.1/CAP Specification of management functions.

FMT_MSA.1.1/CAP-1	The TSF shall enforce the [assignment: *SFP.CAP*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of the specified kernel object*] to [assignment: *R.OWNER, R.SVISITOR*] **by *OP.GRANT, OP.REVOKE***.
FMT_MSA.1.1/CAP-2	The TSF shall enforce the [assignment: *SFP.CAP*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of the specified kernel object*] to [assignment: *R.VISITOR*] **by *OP.REJECT***.

6.2.2.4 FMT_MSA.3/CAP Static attribute initialisation

Hierarchical to:	No other components.
Dependencies:	FMT_MSA.1/CAP Management of security attributes
	FMT_SMR.1 Security roles.

FMT_MSA.3.1	The TSF shall enforce the [assignment: *SFP.CAP*] to provide [selection, choose one of: restrictive] default values for security attributes that are used to enforce the SFP.
FMT_MSA.3.2	The TSF shall allow the [assignment: *R.OWNER, R.SVISITOR*] to specify alternative initial values to override the default values when an object or information is created.

6.2.2.5 FMT_SMF.1/CAP Specification of management functions

Hierarchical to:	No other components.
Dependencies:	No dependencies.

FMT_SMF.1.1	The TSF shall be capable of performing the following management functions: [assignment: *OP.NEWCAP, OP.GRANT, OP.REVOKE, OP.REJECT*].

6.2.3 Information Flow Control and Residual Information Removal

6.2.3.1 FDP_IFC.1/EBB Subset information flow control

Hierarchical to:	No other components.
Dependencies:	FDP_IFF.1/EBB Simple security attributes.

FDP_IFC.1.1 The TSF shall enforce the [assignment: *SFP.EBB*] on [assignment:

- *Subject: S.PROC*
- *Objects: OB.SERVERPOOL, OB.REQUEST*
- *Operations: OP.CALL (on OB.SERVERPOOL) and OP.REPLY (on OB.REQUEST)*

].

6.2.3.2 FDP_IFF.1/EBB Simple security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/EBB Subset information flow control
	FMT_MSA.3/EBB Static attribute initialisation.

FDP_IFF.1.1 The TSF shall enforce the [assignment: *SFP.EBB*] based on the following types of subject and information security attributes: [assignment:

- *The subjects S.PROC and OB.SERVERPOOL and OB.REQUEST with security attributes ATTR.OB.OWNER and ATTR.OB.SVISITOR.*

].

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: [assignment:

- *OP.CALL operation delivers message from one subject to another iff the caller subject has access to an OB.SERVERPOOL, and the recipient acts as the R.OWNER or R.SVISITOR of the specified OB.SERVERPOOL.*
- *OP.REPLY operation delivers message from one subject to another iff the replier subject has access to an OB.REQUEST and acts as the R.OWNER or R.SVISITOR of it, and the recipient holds the access to a related OB.SERVERPOOL.*

].

FDP_IFF.1.3 The TSF shall enforce the [assignment: *None*].

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: [assignment: *None*].

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: [assignment:

- *If the message length exceeds the length threshold, the OP.CALL operation will be denied.*
- *If a subject calls OP.REPLY, which acts as the R.OWNER or the R.SVISITOR of an OB.REQUEST object, while the caller subject associated with this OB.REQUEST object is not waiting for a reply, the OP.REPLY operation will be denied.*

].

6.2.3.3 FMT_MSA.1/EBB Management of security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/EBB Subset information flow control
	FMT_SMR.1 Security roles
	FMT_SMF.1/EBB Specification of management functions.

FMT_MSA.1.1/EBB-1 The TSF shall enforce the [assignment: *SFP.EBB*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of OB.SERVERPOOL*] to [assignment: *R.OWNER, R.SVISITOR*] **by *OP.GRANT, OP.REVOKE***.

FMT_MSA.1.1/EBB-2 The TSF shall enforce the [assignment: *SFP.EBB*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of OB.SERVERPOOL*] to [assignment: *R.VISITOR*] **by *OP.REJECT***.

FMT_MSA.1.1/EBB-3 The TSF shall enforce the [assignment: *SFP.EBB*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of OB.REQUEST*] to [assignment: *None*].

6.2.3.4 FMT_MSA.3/EBB Static attribute initialisation

Hierarchical to:	No other components.
Dependencies:	FMT_MSA.1/EBB Management of security attributes
	FMT_SMR.1 Security roles.

FMT_MSA.3.1 The TSF shall enforce the [assignment: *SFP.EBB*] to provide [selection, choose one of: restrictive] default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the [assignment: *R.OWNER, R.SVISITOR*] to specify alternative initial values to override the default values when an object or information is created.

6.2.3.5 FMT_SMF.1/EBB Specification of management functions

Hierarchical to:	No other components.
Dependencies:	No dependencies.

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

- *Management of security attributes ATTR.OB.PERM of OB.SERVERPOOL via OP.NEWCAP, OP.GRANT, OP.REVOKE, OP.REJECT*
- *Management of signal masking policy via OP.SETCPUMASK and OP.SETPRIOMASK on OB.SERVERPOOL*

].

Application Note a: *OB.SERVERPOOL* can be set to process the requests only from the threads run on the specific CPUs. *OP.SETCPUMASK* sets that set of CPUs.
Application Note b: *OB.SERVERPOOL* can be set to process the requests only from the threads with specific priorities. *OP.SETPRIOMASK* sets that set of priorities.
Application Note c: *OB.REQUEST* cannot be created or granted by users. It is only created inside the kernel when an *OP.CALL* is performed on an *OB.SERVERPOOL*, and the newly created *OB.REQUEST* has the same *R.OWNER* of the *OB.SERVERPOOL*.

6.2.3.6 FDP_IFC.1/ASY Subset information flow control

Hierarchical to:	No other components.
Dependencies:	FDP_IFF.1/ASY Simple security attributes.

FDP_IFC.1.1 The TSF shall enforce the [assignment: *SFP.ASY*] on [assignment:

- *Subject: S.PROC*
- *Object: OB.CHANNEL*
- *Operations: OP.SEND message (to OB.CHANNEL) and OP.RECEIVE message (from OB.CHANNEL).*

].

Application Note: To invoke the operation *OP.SEND* (or *OP.RECEIVE*), an S.PROC should have the *P.SEND* (or *P.RECEIVE*) right.

6.2.3.7 FDP_IFF.1/ASY Simple security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/ASY Subset information flow control
	FMT_MSA.3/ASY Static attribute initialisation.

FDP_IFF.1.1 The TSF shall enforce the [assignment: *SFP.ASY*] based on the following types of subject and information security attributes: [assignment:

- *S.PROC and OB.CHANNEL with security attribute ATTR.OB.PERM.*

].

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: [assignment:

- *OP.SEND operation delivers message to an OB.CHANNEL if caller subject has access to an OB.CHANNEL with the presence of P.SEND right in its ATTR.OB.PERM and the length of message is within the allowed buffer length.*
- *OP.RECEIVE operation grabs message from an OB.CHANNEL if caller subject has access to an OB.CHANNEL with the presence of P.RECEIVE right in its ATTR.OB.PERM.*

].

FDP_IFF.1.3 The TSF shall enforce the [assignment: *None*].

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: [assignment: *None*].

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: [assignment: *None*].

6.2.3.8 FMT_MSA.1/ASY Management of security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/ASY Subset information flow control
	FMT_SMR.1 Security roles
	FMT_SMF.1/ASY Specification of management functions.

FMT_MSA.1.1/ASY-1 The TSF shall enforce the [assignment: *SFP.ASY*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of OB.CHANNEL*] to [assignment: *R.OWNER, R.SVISITOR*] **by *OP.GRANT, OP.REVOKE***.

FMT_MSA.1.1/ASY-2 The TSF shall enforce the [assignment: *SFP.ASY*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of OB.CHANNEL*] to [assignment: *R.VISITOR*] **by *OP.REJECT***.

6.2.3.9 FMT_MSA.3/ASY Static attribute initialisation

Hierarchical to:	No other components.
Dependencies:	FMT_MSA.1/ASY Management of security attributes
	FMT_SMR.1 Security roles.

FMT_MSA.3.1 The TSF shall enforce the [assignment: *SFP.ASY*] to provide [selection, choose one of: restrictive] default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the [assignment: *R.OWNER, R.SVISITOR*] to specify alternative initial values to override the default values when an object or information is created.

6.2.3.10 FMT_SMF.1/ASY Specification of management functions

Hierarchical to: No other components.
Dependencies: No dependencies.

FMT_SMF.1.1 The TSF shall be capable of performing the following management functions: [assignment: *Management of security attributes ATTR.OB.PERM of OB.CHANNEL via OP.NEWCAP, OP.GRANT, OP.REVOKE and OP.REJECT*].

6.2.3.11 FDP_RIP.1 Subset residual information protection

Hierarchical to: No other components.
Dependencies: No dependencies.

FDP_RIP.1.1 The TSF shall ensure that any previous information content of a resource is made unavailable upon the [selection: allocation of the resource to] the following objects: [assignment: *OB.CNODE, OB.THREAD, OB.CHANNEL, OB.SERVERPOOL, OB.REQUEST, OB.VSPACE, OB.PMEM, OB.SYSCTRL, OB.IRQCTRL*].

6.2.4 Memory Management

6.2.4.1 FDP_ACC.1/MEM Subset access control

Hierarchical to: No other components.
Dependencies: FDP_ACF.1/MEM Security attribute based access control.

FDP_ACC.1.1 The TSF shall enforce the [assignment: *SFP.MEM*] on [assignment:

- *Subjects: S.PROC*
- *Objects: physical memory (OB.PMEM) and virtual memory address space (OB.VSPACE)*
- *Operations: OP.MAP, OP.UNMAP, OP.REGISTER_TO_KERNEL, OP.FREE, OP.PLUGIN*

].

6.2.4.2 FDP_ACF.1/MEM Security attribute based access control

Hierarchical to: No other components.
Dependencies: FDP_ACC.1/MEM Subset access control
FMT_MSA.3/MEM Static attribute initialisation.

FDP_ACF.1.1 The TSF shall enforce the [assignment: *SFP.MEM*] to objects based on the following: [assignment:

- *Subject: S.PROC with security attribute: ATTR.PROC.ID*
- *Object: physical memory (OB.PMEM) with security attributes: ATTR.OB.SVISITOR, ATTR.OB.OWNER and ATTR.PMEM.STATUS*
- *Object: virtual memory address space (OB.VSPACE) with security attributes: ATTR.OB.SVISITOR, ATTR.OB.OWNER and ATTR.OB.PERM*

].

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [assignment:

- *S.PROC can OP.MAP an OB.PMEM into an OB.VSPACE iff*
  - *the S.PROC is the R.OWNER or a R.SVISITOR of the OB.PMEM*
  - *the S.PROC has P.MAP right according to the ATTR.OB.PERM of the OB.VSPACE;*
  - *the ATTR.PMEM.STATUS of the OB.PMEM is MAPPED, MAP_ONLY or FREE.*

  The ATTR.PMEM.STATUS of the OB.PMEM will become MAPPED after the operation if the origin ATTR.PMEM.STATUS is MAPPED or FREE.

  The ATTR.PMEM.STATUS of the OB.PMEM is still MAP_ONLY after the operation if the origin ATTR.PMEM.STATUS is MAP_ONLY.
- *S.PROC can OP.UNMAP a virtual region in an OB.VSPACE iff the S.PROC has P.UNMAP right according to the ATTR.OB.PERM of the OB.VSPACE.*
- *S.PROC can OP.REGISTER_TO_KERNEL on an OB.PMEM iff*
  - *the S.PROC is the R.OWNER or a R.SVISITOR of the OB.PMEM*
  - *the ATTR.PMEM.STATUS of the OB.PMEM is FREE.*

  The ATTR.PMEM.STATUS of the OB.PMEM will become KERNEL after the operation.
- *S.PROC can OP.FREE an OB.PMEM iff*
  - *the S.PROC is the R.OWNER or a R.SVISITOR of the OB.PMEM*
  - *the ATTR.PMEM.STATUS of the OB.PMEM is KERNEL.*
  - *the corresponding physical memory is safe to be released, i.e. it is not mapped by any user processes, and it is not used for any kernel object or other kernel data.*

  The ATTR.PMEM.STATUS of the OB.PMEM will become FREE after the operation.
- *S.PROC can OP.PLUGIN an OB.PMEM iff*
  - *the S.PROC is the R.OWNER or a R.SVISITOR of the OB.PMEM*
  - *the ATTR.PMEM.STATUS of the OB.PMEM is VOID.*

  The ATTR.PMEM.STATUS of the OB.PMEM will become MAP_ONLY after the operation.

].

FDP_ACF.1.3 The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: [assignment: *None*].

FDP_ACF.1.4 The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [assignment: *None*].

6.2.4.3 FMT_MSA.1/MEM Management of security attributes

Hierarchical to:	No other components.
Dependencies:	FDP_IFC.1/MEM Subset information flow control
	FMT_SMR.1 Security roles
	FMT_SMF.1/MEM Specification of management functions.

FMT_MSA.1.1/MEM-1 The TSF shall enforce the [assignment: *SFP.MEM*] to restrict the ability to [selection: modify] the security attributes [assignment: *ATTR.OB.PERM of any R.VISITOR of an OB.VSPACE*] to [assignment: *R.OWNER, R.SVISITOR*] by ***OP.GRANT*, *OP.REVOKE***.

FMT_MSA.1.1/MEM-2 The TSF shall enforce the [assignment: *SFP.MEM*] to restrict the ability to [selection: clear] the security attributes [assignment: *ATTR.OB.PERM of any R.VISITOR of an OB.VSPACE*] to [assignment: the *R.VISITOR* itself] by ***OP.REJECT***.

6.2.4.4 FMT_MSA.3/MEM Static attribute initialisation

Hierarchical to:	No other components.
Dependencies:	FMT_MSA.1/MEM Management of security attributes
	FMT_SMR.1 Security roles.

FMT_MSA.3.1 The TSF shall enforce the [assignment: *SFP.MEM*] to provide [selection: restrictive] default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2 The TSF shall allow the [assignment:*R.OWNER, R.SVISITOR*] to specify alternative initial values to override the default values when an object or information is created.

Application Note: All base *OB.PMEM*s are created by TSF during initialization. *R.OWNER* and *R.SVISITOR*s can only split an *OB.PMEM* into two sub *OB.PMEM*s (with new *R.OWNER*s) iff the *OB.PMEM* is FREE or VOID. And they can not specify alternative values for *ATTR.PMEM.STATUS* of the sub *OB.PMEM*s.

6.2.4.5 FMT_SMF.1/MEM Specification of management functions

Hierarchical to:	No other components.
Dependencies:	No dependencies.

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

- *OP.GRANT: Grant virtual memory address space (OB.VSPACE).*
- *OP.REVOKE: Revoke virtual memory address space (OB.VSPACE).*
- *OP.REJECT: Reject virtual memory address space (OB.VSPACE).*
- *OP.NEWCAP: Split physical memory region (OB.PMEM) and change R.OWNER.*

].

6.2.5 Thread Management

6.2.5.1 FRU_PRS.1 Limited priority of service

Hierarchical to:	No other components.
Dependencies:	No other components.

FRU_PRS.1.1 The TSF shall assign a priority to each subject in the TSF.

FRU_PRS.1.2 The TSF shall ensure that each access to [assignment: *CPU time*] shall be mediated on the basis of the subjects assigned priority.

6.2.6 Platform Attestation

6.2.6.1 FAU_SAS.1 Audit storage

Hierarchical to:	No other components.
Dependencies:	No other components.

FAU_SAS.1.1 The TSF shall provide the capability to identify [assignment: *the underlying platforms*] and store [assignment: *the platform identification data*] in the [assignment: *OB.SYSCTRL*].

Application Note: Platform attestation is the means for the system integrator to make sure that the platform is allowed to operate with the TOE.

6.3 Security Functional Requirements Rationale

6.3.1 Security Objectives for the TOE

O.CONFIDENTIALITY The following requirements contribute to fulfill the objective:

- FDP_ACC.1/CAP, FDP_ACF.1/CAP, FDP_ACC.1/MEM and FDP_ACF.1/MEM state the access control measures based on *SFP.CAP* and *SFP.MEM*, which establish capability-based access control policy to kernel objects and memory-based access control policy to memory regions, respectively. Thus keeps kernel objects and memory regions confidential and out of the reach of a third party.
- FDP_IFC.1/EBB, FDP_IFF.1/EBB, FDP_IFC.1/ASY, FDP_IFF.1/ASY, state the information flow control policies based on *SFP.EBB* and *SFP.ASY*, which ensure an inter process communication

channel between the sender and the receiver, preventing a third party to read information transmitted through the communication channel.

- FMT_MSA.1/CAP, FMT_MSA.3/CAP, FMT_SMF.1/CAP, FMT_MSA.1/MEM, FMT_MSA.3/MEM, FMT_SMF.1/MEM, FMT_MSA.1/EBB, FMT_MSA.3/EBB, FMT_SMF.1/EBB, FMT_MSA.1/ASY, FMT_MSA.3/ASY, FMT_SMF.1/ASY establish the management functions available to users to enforce *SFP.CAP*, *SFP.MEM*, *SFP.EBB* and *SFP.ASY*, thus preventing disclosure of data based on the above mentioned SFPs.
- FDP_RIP.1 states resource clean up policy to prevent disclosure of data upon allocation of kernel objects.

O.ACCESS_CONTROL The following requirements contribute to fulfill the objective:

- FIA_ATD.1 enforces the management of the user identity and properties as security attributes, which then become TSF data, as an input for access control functions.
- FIA_UID.2 requires the identification of the user before any action, thus allowing the access and services and data, and their processing, to authorized users only.
- FIA_USB.1 enforces the association of the user identity to the active entity that acts on behalf of the user and to check that this is a valid identity. It is the starting point to ensure that only an authorized user accesses and processes services and data.
- FMT_SMR.1 establishes roles to be used for effective access control and process isolation, used to support management functions for *SFP.CAP*, *SFP.MEM*, *SFP.EBB* and *SFP.ASY*.
- FDP_ACC.1/CAP, FDP_ACF.1/CAP, FDP_ACC.1/MEM and FDP_ACF.1/MEM state the access control measures based on *SFP.CAP* and *SFP.MEM*, which establish capability-based access control policy to kernel objects and memory-based access control policy to memory regions, respectively. Thus keeps kernel objects and memory regions inaccessible to unauthorized users.
- FDP_IFC.1/EBB, FDP_IFF.1/EBB, FDP_IFC.1/ASY, FDP_IFF.1/ASY, state the information flow control policies based on *SFP.EBB* and *SFP.ASY*, which ensure an inter process communication channel between the sender and the receiver, preventing an aunauthorized user to access a communication channel.
- FMT_MSA.1/CAP, FMT_MSA.3/CAP, FMT_SMF.1/CAP, FMT_MSA.1/MEM, FMT_MSA.3/MEM, FMT_SMF.1/MEM, FMT_MSA.1/EBB, FMT_MSA.3/EBB, FMT_SMF.1/EBB, FMT_MSA.1/ASY, FMT_MSA.3/ASY, FMT_SMF.1/ASY establish the management functions available to users to enforce *SFP.CAP*, *SFP.MEM*, *SFP.EBB* and *SFP.ASY*, thus preventing unauthorized access to data, memory regions or communication channels based on the above mentioned SFPs.

O.SAFE_SECURE_STATE

- FPT_FLS.1 states that the TOE has to reach a secure state upon initialization or device binding failure.

O.PRIORITY

- FRU_PRS.1 states that the TOE has to manage the usage of resources based on a priority-based queue.

O.ATTESTATION

- FAU_SAS.1 states that the TOE has the means for the system integrator to univocally check that the platform is allowed to be integrated with the TOE.

6.3.2 Coverage and Sufficiency

This part provides the rationale for the internal consistency and completeness of the SFRs defined in this Security Target.

Security Objectives	Security Functional Requirements
O.CONFIDENTIALITY	FDP_ACC.1/CAP, FDP_ACC.1/MEM, FDP_ACF.1/CAP, FDP_ACF.1/MEM, FDP_IFC.1/EBB, FDP_IFC.1/ASY, FDP_IFF.1/EBB, FDP_IFF.1/ASY, FDP_RIP.1, FMT_MSA.1/EBB, FMT_MSA.1/ASY, FMT_MSA.1/CAP, FMT_MSA.1/MEM, FMT_MSA.3/EBB, FMT_MSA.3/ASY, FMT_MSA.3/CAP, FMT_MSA.3/MEM, FMT_SMF.1/EBB, FMT_SMF.1/ASY, FMT_SMF.1/CAP, FMT_SMF.1/MEM.
O.ACCESS_CONTROL	FIA_ATD.1, FIA_UID.2, FIA_USB.1, FDP_ACC.1/CAP, FDP_ACC.1/MEM, FDP_ACF.1/CAP, FDP_ACF.1/MEM, FDP_IFC.1/EBB, FDP_IFC.1/ASY, FDP_IFF.1/EBB, FDP_IFF.1/ASY, FMT_MSA.1/EBB, FMT_MSA.1/ASY, FMT_MSA.1/CAP, FMT_MSA.1/MEM, FMT_MSA.3/EBB, FMT_MSA.3/ASY, FMT_MSA.3/CAP, FMT_MSA.3/MEM, FMT_SMF.1/EBB, FMT_SMF.1/ASY, FMT_SMF.1/CAP, FMT_SMF.1/MEM, FMT_SMR.1.
O.SAFE_SECURE_STATE	FPT_FLS.1
O.PRIORITY	FRU_PRS.1
O.ATTESTATION	FAU_SAS.1

Table 5: Security Objectives and SFRs - Coverage

Security Functional Requirements	Objectives
FIA_ATD.1	O.ACCESS_CONTROL
FIA_UID.2	O.ACCESS_CONTROL
FIA_USB.1	O.ACCESS_CONTROL
FDP_IFC.1/EBB	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_IFC.1/ASY	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_IFF.1/EBB	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_IFF.1/ASY	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_ACC.1/CAP	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_ACC.1/MEM	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_ACF.1/CAP	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_ACF.1/MEM	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FDP_RIP.1	O.CONFIDENTILITY
FMT_MSA.1/EBB	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.1/ASY	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.1/CAP	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.1/MEM	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.3/EBB	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.3/ASY	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.3/CAP	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_MSA.3/MEM	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_SMF.1/EBB	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_SMF.1/ASY	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_SMF.1/CAP	O.CONFIDENTIALITY, O.ACCESS_CONTROL
FMT_SMF.1/MEM	O.CONFIDENTIALITY, O.ACCESS_CONTROL

Security Functional Requirements	Objectives
FMT_SMR.1	O.ACCESS_CONTROL
FPT_FLS.1	O.SAFE_SECURE_STATE
FRU_PRS.1	O.PRIORITY
FAU_SAS.1	O.ATTESTATION

Table 6: Security Objectives and SFRs - Sufficiency

6.3.3 Dependencies

The following table presents the dependencies between different security functional components.

SFRs	CC Dependencies	Satisfied Dependencies
FIA_ATD.1	No Dependencies	
FIA_UID.2	No Dependencies	
FIA_USB.1	FIA_ATD.1	FIA_ATD.1
FDP_ACC.1/MEM	FDP_ACF.1/MEM	FDP_ACF.1/MEM
FDP_ACC.1/CAP	FDP_ACF.1/CAP	FDP_ACF.1/CAP
FDP_ACF.1/MEM	FDP_ACC.1/MEM and FMT_MSA.3/MEM	FDP_ACC.1/MEM FMT_MSA.3/MEM
FDP_ACF.1/CAP	FDP_ACC.1/CAP and FMT_MSA.3/CAP	FDP_ACC.1/CAP FMT_MSA.3/CAP
FDP_IFC.1/EBB	FDP_IFF.1/EBB	FDP_IFF.1/EBB
FDP_IFC.1/ASY	FDP_IFF.1/ASY	FDP_IFF.1/ASY
FDP_IFF.1/EBB	FDP_IFC.1/EBB and FMT_MSA.3/EBB	FDP_IFC.1/EBB FMT_MSA.3/EBB
FDP_IFF.1/ASY	FDP_IFC.1 and FMT_MSA.3	FDP_IFC.1/ASY FMT_MSA.3/ASY
FDP_RIP.1	No Dependencies	
FMT_MSA.1/EBB	FDP_ACC.1/EBB or FDP_IFC.1/EBB and FMT_SMF.1/EBB and FMT_SMR.1	FDP_IFC.1/EBB FMT_SMF.1/EBB FMT_SMR.1
FMT_MSA.1/ASY	FDP_ACC.1/ASY or FDP_IFC.1/ASY and FMT_SMF.1/ASY and FMT_SMR.1	FDP_IFC.1/ASY FMT_SMF.1/ASY FMT_SMR.1
FMT_MSA.1/MEM	FDP_ACC.1/MEM or FDP_IFC.1/MEM and FMT_SMF.1/MEM and FMT_SMR.1	FDP_ACC.1/MEM FMT_SMF.1/MEM FMT_SMR.1
FMT_MSA.1/CAP	FDP_ACC.1/CAP or FDP_IFC.1/CAP and FMT_SMF.1/CAP and FMT_SMR.1	FDP_ACC.1/CAP FMT_SMF.1/CAP FMT_SMR.1
FMT_MSA.3/EBB	FMT_MSA.1/EBB and FMT_SMR.1	FMT_MSA.1/EBB FMT_SMR.1
FMT_MSA.3/ASY	FMT_MSA.1/ASY and FMT_SMR.1	FMT_MSA.1/ASY FMT_SMR.1
FMT_MSA.3/MEM	FMT_MSA.1/MEM and FMT_SMR.1	FMT_MSA.1/MEM FMT_SMR.1

SFRs	CC Dependencies	Satisfied Dependencies
FMT_MSA.3/CAP	FMT_MSA.1/CAP and FMT_SMR.1	FMT_MSA.1/CAP FMT_SMR.1
FMT_SMF.1	No Dependencies	
FPT_FLS.1	No Dependencies	
FRU_PRS.1	No Dependencies	
FAU_SAS.1	No Dependencies	

Table 7: SFRs Dependencies

6.4 Security Assurance Requirements

This ST follows a set of Security Assurance Requirements (SARs) for the TOE. It consists of the EAL 5 package augmented with ALC_FLR.1.

6.5 Security Assurance Requirements Rationale

The Evaluation Assurance Level 5 has been chosen to commensurate with the threat environment that is experienced by typical consumers of the TOE. The assurance level defined in this ST consists of the predefined assurance package EAL 5 with the augmentation ALC_FLR.1.

7 TOE Summary Specification

For every security function a short identifier is specified in brackets to allow direct referencing to single items in other documents.

The TOE performs the following security functions:

- User Identification and TSF Protection
- Capability-based Access Control
- Information Flow Control and Residual Information Removal
- Memory Management
- Thread Management
- Platform attestation

7.1 User Identification and TSF Protection

7.1.1 Identification and Authentication

The TOE does not identify physical users, only processes are identified by the TOE. Processes can be viewed as the users and subjects at the same time.

Each process is identified by the process identifier *ATTR.PROC.ID*. The TOE maintains the security attributes for each process that take the actions on behalf of a specified process.

Each kernel object has an associated owner process that is identified in *ATTR.OB.OWNER*, and may also allow supervised access to other processes by *ATTR.OB.SVISITOR*.

The functions authorized to each process are mapped in *ATTR.OB.PERM*.

The attributes mentioned above are bound to the corresponding S.PROC as unique process identifier, kernel object owner, process with supervised access and/or authorized operations, where applicable.

7.1.2 Protection of TSF

If the TSF is identified as a failure state in its initialization and runtime, it will *panic*, stop the TOE, and enters the secure halt state. This status indicates a secure status when a failure condition is faced with. To reenter the secure state, the TOE should be restarted.

The TOE utilizes *BUG_ON* mechanism to check state consistency of the TOE. The checks are implemented throughout all the key paths in the code, to check if key logical conditions are meet. If any memory error or glitch causes logical inconsistency and get caught by the *BUG_ON*s, the kernel will *panic* immediately and halts.

7.1.3 SFR Summary

This security function covers the following SFRs: (FIA_ATD.1, FIA_UID.2, FIA_USB.1, FPT_FLS.1).

7.2 Capability-based Access Control

7.2.1 Capability

7.2.1.1 Discretionary Access Control

The TOE enforces capability-based access control as the Discretionary Access Control (DAC) mechanism to protect the confidentiality of the kernel objects.

A *capability* is an unforgeable token of authority. Each kernel object is associated with a *capability*. To access a kernel object, a process should first have the corresponding *capability*. A *capability* specifies the rights a process is authorized to access the kernel object.

The *capability* is assigned to the kernel objects when it is created by means of *OP.NEWCAP*.

Each process has a collection of capabilities called CNode (*OB.CNODE*), which describes all the kernel objects it is authorized to access. One CNode corresponds to and only to one process.

Each kernel object has one owner (i.e., a process bound to the role *R.OWNER*), and may have multiple visitors. The owner can grant the *capability* of the kernel object to other processes to allow other processes to access it by means of *OP.GRANT*. The owner can revoke the authority anytime by means of *OP.REVOKE*. The process which accepts the *capability* is called the visitor of the kernel object bound to the role *R.VISITOR*, and may also reject granted capabilities by means of *OP.REJECT*. The visitor cannot grant the *capability* to other processes.

In addition, the TOE implements a supervised access mode bound to the role *R.SVISITOR*, where an owner of a CNode can grant its CNode to other processes, and allows other processes to gain full access its *capabilities* on behalf of it. In this scenario, those processes holds the granted CNode are called supervised visitors (*R.SVISITOR*) to the *capabilities* in the granted CNode.

The rights of the kernel objects are stored in the corresponding capabilities. In the following, the ST author refers the *capability* to represent the rights.

7.2.1.2 DAC Attributes

The DAC attributes of the kernel objects consist of the following parts:

- *ATTR.OB.OWNER*: The owner of the kernel object. It can give the authorization with the specified rights to other processes.
- *ATTR.OB.SVISITOR*: A set which records the supervised visitors of the kernel object. A supervised visitor is added to the set when the owner of the kernel object grants its CNode to another process. At this point the process with granted CNode becomes a supervised visitor to the kernel object.
- *ATTR.OB.PERM*: A mapping from the processes to the rights/*capabilities*. It specifies which processes can do what operations on the kernel object. The owner and the supervised visitors can modify *ATTR.OB.PERM*.

  Actually, the TOE does not use a mapping to store the *ATTR.OB.PERM*. The permissions reside in the CNode of individual processes. Inside the CNode, there are two parts, storing two kinds of capabilities separately:

  - CNode.Own: A set of capabilities it owns.
  - CNode.Grant: A set of capabilities it is granted.

  A process grants the *capability* of the kernel object to another process by adding the *capability* to the CNode.Grant of the target process.

  A process revokes the *capability* of the kernel object from the specified process by removing the *capability* from the CNode.Grant of the target process.

7.2.1.3 DAC Enforcement Algorithm

The TSF enforces the DAC policy to kernel objects based on the *capability*. The TOE uses the following rules to determine whether a request to access the kernel object is permissible. To access the kernel object, a process should show its *capability* list.

- Check the validity of the *capability*: The *capability* must be a valid element inside CNode of the requester. Otherwise the request is refused.
- Check the rights specified in the *capability*: The operations to take on the kernel object must be allowed by the rights described in the *capability*. Otherwise the request is refused.

7.2.2 Security Management in Capability

The owner and the supervised visitor of a kernel objects manages the security attributes. When a kernel object is created, the owner and the supervised visitor specifies the initial rights to access it. The initial rights cannot be modified afterwards, and the rights granted to other processes must be less than the initial rights.

A process can grant or revoke kernel objects to other processes only if it has the kernel objects' corresponding *capabilities* in its CNode.Own, or it has the supervised access to those *capabilities*.

7.2.3 SFR Summary

This security function covers the following SFRs: (FDP_ACC.1/CAP, FDP_ACF.1/CAP, FMT_MSA.1/CAP, FMT_MSA.3/CAP, FMT_SMF.1/CAP, FMT_SMR.1).

7.3 Information Flow Control and Residual Information Removal

7.3.1 IPC Information Control Policy

The TOE implements *IPC Information Control Policy* to control information flows associated with inter process communications, namely EBBCall and Async IPC. In general, the TOE enforces that all the IPC related objects (*OB.SERVERPOOL*, *OB.CHANNEL* and *OB.REQUEST*) are under the control of CAP, and only the authorized subjects with correct roles can initiate IPC operations to transfer data from one process to another. Any unauthorized attempt to transmit information between subjects via IPC will be immediately denied.

7.3.1.1 IPC Information Control Policy on EBBCall

For EBBCall, the TOE manages a group of *OB.THREAD*s as receivers via *OB.SERVERPOOL*, those threads all belong to a S.PROC, which also acts as the *R.OWNER* of this *OB.SERVERPOOL*. The *R.OWNER* and the *R.SVISITOR* of the *OB.SERVERPOOL* can configure the receiver threads in the *OB.SERVERPOOL*. Under the control of CAP, only a S.PROC acts as the *R.VISITOR* of an *OB.SERVERPOOL* is allowed to send data to receiver threads via *OP.CALL* operation.

Along with receiving a call from a valid *OB.SERVERPOOL*, the *R.OWNER* and the *R.SVISITOR* of the *OB.SERVERPOOL* also gains possession of an *OB.REQUEST* object. *OB.REQUEST* is used for identifying the origin of the call (as S.PROC), the *R.OWNER* and the *R.SVISITOR* can invoke *OP.REPLY* on it to send a reply message back to the caller. Under the control of CAP, *OB.REQUEST* will not have any *R.VISITOR*. The *R.OWNER* and the *R.SVISITOR* can initiate a reply procedure on it which sends data back to the origin S.PROC.

In addition, *OB.REQUEST* keeps track of the caller's state, only the origin S.PROC which is actually waiting for a reply can accept an *OP.REPLY*. Any attempt to invoke *OP.REPLY* to an *OB.REQUEST* which holds an origin S.PROC with an invalid state will be denied.

In the event that the message length exceeds the length threshold, the *OP.CALL* operation will be denied.

7.3.1.2 IPC Information Control Policy on Async IPC

For Async IPC, the TOE uses *OB.CHANNEL* to incorporate S.PROCs that act as senders and receivers. An *OB.CHANNEL* has attribute *ATTR.OB.PERM*, which consists of *P.SEND* right and/or *P.RECEIVE* right. Under the control of CAP, for all S.PROCs with access (*R.OWNER*, *R.VISITOR* and *R.SVISITOR*) to a particular *OB.CHANNEL*, only the S.PROC with *P.RECEIVE* right can invoke *OP.RECEIVE* on the *OB.CHANNEL* to receive messages, and only the S.PROC with *P.SEND* right can send messages to the *OB.CHANNEL*. Any attempt tries to receive or send data on an unauthorized *OB.CHANNEL* will be denied.

7.3.2 IPC Security Management

7.3.2.1 Security Management on EBBCall

The basic security attributes of EBBCall are managed under CAP, which controls the accessibility of *OB.SERVERPOOL*, *OB.REQUEST* based on roles (*R.OWNER R.SVISITOR R.VISITOR*). In addition, *OB.REQUEST* is statically configured to be only accessible by *R.OWNER*, i.e, it cannot be granted to other S.PROC thus will not have any *R.VISITOR*.

7.3.2.2 Security Management on Async IPC

The TOE uses *OB.CHANNEL* as medium to implement Async IPC, and since *OB.CHANNEL* is managed by capability subsystem, its security management functions obey capability operations. There is no additional rules to limit the management.

7.3.3 Residual Information Removal

Kernel objects which have a dedicated memory region are zeroed in the creation procedure by wiping the data stored in the memory. Specifically, the wiping process is applied over *OB.CNODE*, *OB.THREAD*, *OB.CHANNEL* and *OB.SERVERPOOL*.

The other kernel objects do not have a dedicated memory region. In such cases, the object data is initialized with the corresponding default values, thus the kernel objects do not contain any trace of residual information.

7.3.4 SFR Summary

This security function covers the following SFRs: (FDP_IFC.1/EBB, FDP_IFF.1/EBB, FMT_MSA.1/EBB, FMT_MSA.3/EBB, FMT_SMF.1/EBB, FDP_IFC.1/ASY, FDP_IFF.1/ASY, FMT_MSA.1/ASY, FMT_MSA.3/ASY, FMT_SMF.1/ASY, FMT_SMR.1, FDP_RIP.1).

7.4 Memory Management

7.4.1 Memory Management Policy

7.4.1.1 Capability based Physical Memory Management

TOE uses the kernel object *OB.PMEM* to record the information of physical memory regions. The *R.OWNER* of the physical memory region is a user process that owns the capability of the *OB.PMEM*. There is no *R.VISITOR* role for *OB.PMEM* since *OB.PMEM* is not allowed to be granted.

The user processes, to which the *R.OWNER* grants its *OB.CNODE*, are the *R.SVISITOR* of the *OB.PMEM*. *OB.PMEM* uses a field to record how the corresponding physical memory region is being used. It can be classified into:

- MAP_ONLY: user processes can only do mapping on it, e.g. it is mapped by MMIO device or shared with external system.
- FREE: it is free.
- MAPPED: it is being mapped by user processes.
- KERNEL: it is being used to store kernel data, so user process should not access it.
- VOID: there is no corresponding physical memory, but it can be transformed into MAP_ONLY when add physical memory as hot plugging.

7.4.1.2 Capability based Virtual Memory Management

TOE uses kernel objects *OB.VSPACE* to record the information of virtual memory address spaces. The *R.OWNER* of the virtual memory address space is a user process that owns a capability of an *OB.VSPACE*. And the *R.VISITOR* of the virtual memory address space a user process which a capability of an *OB.VSPACE* has been granted to. The user processes, to which the *R.OWNER* grants its *OB.CNODE*, are the *R.SVISITOR* of the *OB.VSPACE*. All permissions (owned or granted) of an *OB.VSPACE* recorded in capability subsystem are within the *ATTR.OB.PERM* of the virtual memory address space.

7.4.1.3 Syscalls of Memory Management

All syscalls involving memory management in the TOE can be categorised as follows:

- Syscalls that map a physical memory region to user virtual memory address space by means of *OP.MAP*;
- Syscalls that unmap a physical memory region from user virtual memory address space by means of *OP.UNMAP*;
- Syscalls that change an *OB.PMEM*'s *ATTR.PMEM.STATUS*, including:
  - Specify physical memory to store kernel data by means of *OP.REGISTER_TO_KERNEL*.
  - Release specified physical memory from storing kernel data by means of *OP.FREE*.
  - Plug in a physical memory block on a specified region by means of *OP.PLUGIN*.

These kinds of syscalls receive *capability* as a parameter, and check the permissions through capability system. TOE never handles a syscall if:

- The user process provides an *OB.PMEM* that is owned by other process.
- The user process tries to operate an *OB.VSPACE* without corresponding permission (*P.MAP*, *P.UNMAP*).
- It tries to map an *OB.PMEM* that is not FREE, MAP_ONLY or MAPPED.
- It tries to store kernel data into a non-FREE *OB.PMEM*.
- It tries to plug in a physical memory block on the region of a non-VOID *OB.PMEM*.

7.4.2 Security Management in Memory Management

7.4.2.1 Physical Memory Ownership Transfer

There is a default ownership of all *OB.PMEM*s after kernel booting. User processes cannot modify *OB.PMEM*'s owner directly. However, an *OB.PMEM* can be used to create two sub *OB.PMEM*s, and each sub *OB.PMEM* represents a sub physical memory region. Once an *OB.PMEM* has been used to create sub *OB.PMEM*s, the origin *OB.PMEM* becomes unavailable. When a user process creates a sub *OB.PMEM*, it stores the capability of the new *OB.PMEM* in a specified authorized *OB.CNODE*, thus the *R.OWNER* of the physical memory may change. And the user processes, to which the specified *OB.CNODE* has been granted to, are the *R.SVISITOR*s of the new *OB.PMEM*.

7.4.2.2 Virtual Memory Address Space Authorization

When a virtual memory address space is created, it must bind to a *OB.CNODE* with full permissions. Since a virtual memory address is managed by capability system, its security management functions are same as capability operations including *OP.GRANT*, *OP.REVOKE* and *OP.REJECT*. There is no additional rule to limit the authorization.

7.4.3 SFR Summary

This security function covers the following SFRs: (FDP_ACC.1/MEM, FDP_ACF.1/MEM, FMT_MSA.1/MEM, FMT_MSA.3/MEM, FMT_SMF.1/MEM, FMT_SMR.1).

7.5 Thread Management

7.5.1 Kernel Scheduling

In the priority-based FIFO scheduling, ready threads with the same priority are linked in a queue which is sorted by the enqueue order. The scheduler is constructed with multiple queues where each queue stands for a given priority. Upon scheduling, the scheduler finds a non-empty queue with the highest priority, then chooses the thread in the head of the queue to run. Each thread has a timeslice to keep track of the current remaining time on the CPU, which will be decreased by 1 upon each timer tick. When the timeslice is used up, the current thread will be append to the tail of the queue of the corresponding priority queue in scheduler. At this point the scheduler will choose a new thread from its queues (including the current thread) to execute. Note that in the HongMeng, the current thread also gives up the CPU whenever a syscall occurs.

7.5.2 Futex

The Fast Userspace muTEXes (Futex), is a light-weight method for process synchronization in HongMeng. The Futex operation is atomic and involves the memory shared by synchronized processes. When a process enters or exits a race condition, the process first checks the Futex variable to find whether the race happens. If the race happens, the process will be suspended in the kernel with the FUTEX_WAIT call, and the kernel will wake up this process with FUTEX_WAKE. On the contrary, the process will not be trapped in the kernel space if there is no race, and this may improve the kernel efficiency.

7.5.3 SFR Summary

This security function covers the following SFR: (FRU_PRS.1).

7.6 Platform Attestation

The TOE stores unique platform attestation data to be used in its operational state. Specifically, platform identification and allowed configuration values are stored in the kernel object *OB.SYSCTRL*.

7.6.1 SFR Summary

This security function covers the following SFR: (FAU_SAS.1).

8 Abbreviations

CA	Client Application
CC	Common Criteria
CEM	Common Evaluation Methodology
DAC	Discretionary Access Control
EAL	Evaluation Assurance Level
GIC	General Interrupt Controller
IPC	Inter Process Communication
IT	Information Technology
MMU	Memory Management Unit
NVM	Non-Volatile Memory
OS	Operating System
OSP	Organisational Security Policy
PP	Protection Profile
RAM	Random Access Memory
REE	Rich Execution Environment
RFC	Request For Comments
SAR	Security Assurance Requirement
SFP	Security Function Policy
SFR	Security Functional Requirement
SoC	System-on-Chip
SPD	Security Problem Definition
ST	Security Target
TA	Trusted Application
TEE	Trusted Execution Environment
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSFI	TSF Interface
TSS	TOE Summary Specification

9 Terminologies

This section contains definitions of technical terms that are used with a meaningful specific to this document. Terms defined in the [CC1] are not reiterated here, unless stated otherwise.

ARM Trusted Firmware	A firmware providing a reference implementation of secure world software for ARMv7-A and ARMv8-A.
Bootloader	A piece of program that loads and starts the kernel.
Client Application	The applications running in the rich execution environment.
Kirin	A series of integrated circuit chips produced by Huawei HiSilicon.
MAP	The operation to establish the mapping from the specific virtual addresses to specific physical addresses.
MMU	Address translation hardware in the CPU which automatically translates virtual addresses to physical addresses.
Page Table	A data structure recording the mapping relations between virtual addresses and physical addresses.
Panic	A safety measure taken by an operating system's kernel upon detecting an internal fatal error in which it either is unable to safely recover from or cannot have the system continue to run without having a much higher risk of major data loss.
REE	The normal processing environment which runs the user-facing operation system. It is isolated from TEE.
TEE	A secure area of a main processor. It guarantees code and data loaded inside to be protected with respect to confidentiality and integrity .
Trusted Application	The applications run in the trusted execution environment.
TrustZone	A technology providing system-wide hardware isolation for trusted software.
UNMAP	The operation to erase the mapping from the specific virtual addresses to specific physical addresses.
Virtual Memory	A memory management technique that provides an abstracted view of all accessiable memory resources on a machine.

10 References

[CC1]	Common Criteria for Information Technology Security Evaluation Part 1: Introduction and General Model, Version 3.1, Revision 5, April 2017, CCMB-2017-04-001.
[CC2]	Common Criteria for Information Technology Security Evaluation Part 2: Security Functional Requirements, Version 3.1, Revision 5, April 2017, CCMB-2017-04-002.
[CC3]	Common Criteria for Information Technology Security Evaluation Part 3: Security Assurance Requirements, Version 3.1, Revision 5, April 2017, CCMB-2017-04-003.
[CEM]	Common Methodology for Information Technology Security Evaluation: Evaluation Methodology, Version 3.1, Revision 5, April 2017, CCMB-2017-04-004.
[ARC]	Huawei HongMeng Common Criteria Evaluation ADV_ARC: Security Architecture, Version 1.2, July 2019.
[OPE]	Huawei HongMeng Common Criteria Evaluation AGD_OPE: Operational User Guidance, Version 2.0, July 2019.
[PRE]	Huawei HongMeng Common Criteria Evaluation AGD_PRE: Preparative Procedures, Version 1.4, July 2019.