IBM

SECURITY TARGET LITE

NXP P541G072V0P (JCOP 41 v2.3.1)
Secure Smart Card Controller

Version **1.0**

2007-07-23

IBM Deutschland Entwicklung GmbH
Smart Card Development
Schoenaicher Str. 220
D-71032 Boeblingen
Germany

Table of Contents

1 ST Introduction

This Security Target Lite has been derived from the full Security Target Version 2.13, 1st June, 2007.

1.1 ST Identification

Title:	Security Target Lite – NXP P541G072V0P (JCOP 41 v2.3.1) Secure Smart Card Controller
Version:	1.0
Date:	2007-07-23
Author(s):	IBM Deutschland Entwicklung GmbH
Developer:	IBM Deutschland Entwicklung GmbH Smart Card Development Schoenaicher Str. 220 D-71032 Boeblingen Germany
Product type:	Java Card
TOE name/version:	NXP P541G072V0P (JCOP 41 v2.3.1)[1]
TOE Software:	IBM WECOS R5 JCOP41, v2.3.1
TOE Hardware:	NXP P5CT072V0P Secure Smart Card Controller
CC used [CC]:	Common Criteria for Information Technology Security Evaluation, version 2.3, August 2005 Part 1: Introduction and general model, CCMB-2005-08-001, Part 2: Security functional requirements, CCMB-2005-08-002, Part 3: Security Assurance Requirements, CCMB-2005-08-003.

1.2 ST Overview

This document details the security target (ST) for Java Card JCOP 41, v2.3.1. It is based on the following protection profile:

- Java Card System Protection Profile Collection, Version: 1.0b, August 2003. – Minimal Configuration Protection Profile [JCSPP]

This ST makes claims for formal conformance to this PP, as the ST fulfils all requirements of [JCSPP]. This ST even chooses a hierarchically higher augmentation of EAL4, in comparison to [JCSPP], by selecting ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3.

Furthermore, parts of the security environment, security objectives and IT security requirements were inspired from the draft document: *Protection Profile – Smart Card*

[1] In the following shortly termed JCOP 41.

Native Operating System Draft Version 0.5, Issue April 2004 and from *Protection Profile Machine Readable travel Document with "ICAO Application", Basic Access Control* [PP0017].

The hardware platform NXP P5CT072V0P including all minor configuration options as defined in section 2.2.5 of the Security Target [ST0348] is certified by BSI (BSI-DSZ-CC-0348-2006 [BSI0348]) and is compliant to the following protection profile:

- Smartcard IC Platform Protection Profile, Version 1.0, July 2001 [PP0002]

For this TOE three minor configuration options can be freely chosen during Smartcard Personalization (see section 2.2.5 of the Security Target [ST0348]):

- "MIFARE Emulation = A" in which MIFARE interface is disabled.
- "MIFARE Emulation = B1" in which MIFARE interface is enabled and 1KB MIFARE EEPROM memory is reserved
- "MIFARE Emulation = B4" in which MIFARE interface is enabled and 4KB MIFARE EEPROM memory is reserved

From [PP0002] relevant requirements for the hardware platform were taken. The relevant requirements for the Java Card functionality were taken from [JCSPP].

JCOP 41, v2.3.1 is based on JavaCard 2.2.1 and Global Platform 2.1.1 industry standards. It implements high security mechanisms and supports:

- different communication protocols:
  - T=0
  - T=1
  - T=CL (contact-less)
  - USB 2.0
- cryptographic algorithms and functionality:
  - 3DES
  - AES (Advanced Encryption Standard)
  - RSA
  - SHA-1
  - random number generation

1.3 CC Conformance Claims

The TOE is [CC] part 2 extended by FCS_RND.1, FMT_LIM.1, FMT_LIM.2 and FPT_EMSEC.1.

The TOE is [CC] part 3 conformant, i. e. all assurance components are taken from part 3 [CC]. The Evaluation Assurance Level is **EAL4 augmented by ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3**.

The minimum strength of the TOE security functions is **SOF-high**.

This ST makes claims for formal conformance to this [JCSPP].

2 TOE Description

This part of the document describes the TOE to provide an understanding of its security requirements, and addresses the product type and the general IT features of the TOE.

2.1 TOE Abstract and Definition

The target of evaluation (TOE) is the Java Card NXP P541G072V0P (JCOP 41 v2.3.1) and consists of:

- Smart Card Platform SCP (hardware platform and hardware abstraction layer)
- embedded software (Java Card Virtual Machine, Runtime Environment, Java Card API, Card Manager), and
- native MIFARE application (physically present but logically disabled in minor configuration “MIFARE Emulation = A” and logically enabled in the minor configurations “MIFARE Emulation = B1” and “MIFARE Emulation = B4”. See section 2.2.5 of [ST0348].)

but not software for the application layer (Java applets). This is shown schematically in the following figure:

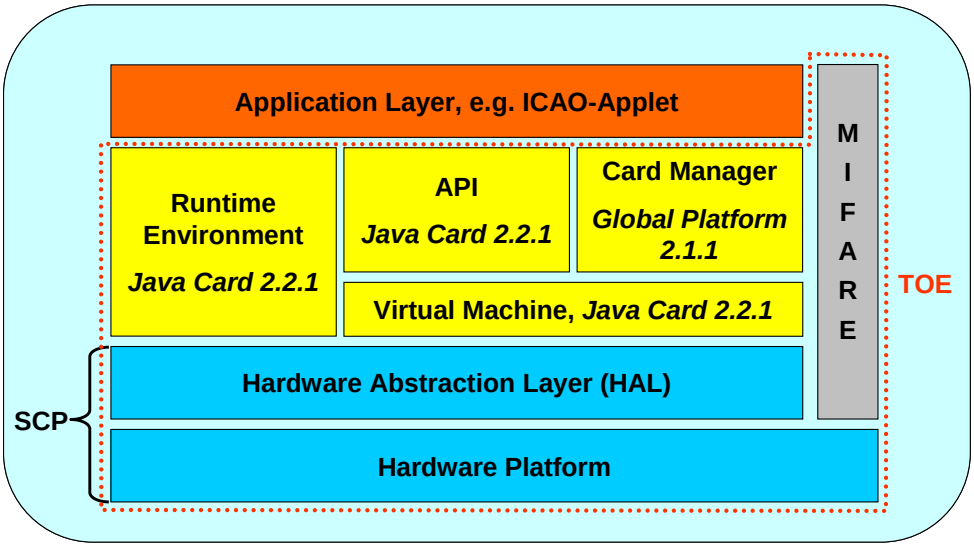


Figure 1: Java Card architechture

The definition of the TOE corresponds to the Minimal Configuration Protection Profile [JCSPP] extended by the Card Manager and the Smart Card Platform. The Smart Card Platform (SCP) consists of the HAL and the Hardware Platform.

The Java Card virtual machine (*JCVM*) is responsible for ensuring language-level security; the *JCRE* provides additional security features for Java Card technology-enabled devices.

The basic runtime security feature imposed by the *JCRE* enforces isolation of *applet*s using an *applet firewall*. It prevents objects created by one *applet* from being used by another *applet* without explicit sharing. This prevents unauthorized access to the fields and methods of *class* instances, as well as the length and contents of arrays.

The *applet firewall* is considered as the most important security feature. It enables complete isolation between *applet*s or controlled communication through additional mechanisms that allow them to share objects when needed. The *JCRE* allows such sharing using the concept of “shareable interface objects” (*SIO*) and **static public** variables. The *JCVM* should ensure that the only way for *applet*s to access any resources are either through the *JCRE* or through the Java Card API (or other vendor-specific APIs). This objective can only be guaranteed if *applet*s are correctly typed (all the “must clauses” imposed in chapter 7 of [JCVM221] on the bytecodes and the correctness of the *CAP* file format are satisfied).

The Card Manager is conformant to the Open Platform Card Specification 2.0.1 [OPCS] and is responsible for the management of applets in the card. No post-issuance loading and deletion of applets is allowed for the present TOE. For more details of the Java card functionality see section 2.3.

The hardware platform, the NXP P5CT072V0P Secure Smart Card Controller has been certified at BSI under registration number BSI-DSZ-CC-0348-2006 to EAL5 augmented by ALC_DVS.2, AVA_MSU.3, and AVA_VLA.4 with all minor configuration options as defined in section 2.2.5 of the Security Target [ST0348]. For this TOE the minor configuration options “MIFARE Emulation = A”, “MIFARE Emulation = B1” and “MIFARE Emulation = B4” can freely be chosen. (see section 2.2.5 of the Security Target [ST0348])

The present evaluation is a composite evaluation using the results of this hardware certification.

The Java card is based on JavaCard 2.2.1 and GlobalPlatform 2.1.1 industry standards. The following features comprise the logical scope of the TOE:

- 6 different communication protocols:
    - ISO 7816 T=1 direct convention
    - ISO 7816 T=0 direct convention
    - ISO 7816 T=1 inverse convention
    - ISO 7816 T=0 inverse convention
    - ISO 14443 T=CL (contact-less)
    - ISO 7816-12 USB (2.0)

- cryptographic algorithms and functionality:
  - 3DES (112 and 168 bit keys) for en-/decryption (CBC and ECB) and signature (MAC) generation and verification
  - AES (Advanced Encryption Standard) with key length of 128, 192, and 256 Bit for en-/decryption (CBC and ECB)
  - RSA (1024 up to 2368 bits keys) for en-/decryption and signature generation and verification
  - SHA-1 hash algorithm
  - random number generation according to class K3 of [AIS 20]
- JavaCard 2.2.1 functionality:
  - Garbage Collection fully implemented with complete memory reclamation incl. compactification
- GlobalPlatform 2.1.1 functionality:
  - CVM Management (Global PIN) fully implemented: all described APDU and API interfaces for this feature are present
  - Secure Channel Protocol (SCP01, and SCP02) is supported
- functionality as defined in the [JCSPP] minimal configuration (i. e. no post-issuance installation and deletion of applets, packages and objects, no RMI, no logical channels, no on-card Bytecode verification), and
- card manager functionality for pre-issuance loading and management of packages and applets.

2.2 TOE Life-Cycle

The life-cycle for this Java Card is based on the general smart card life-cycle defined in [PP0002] and has been adapted to Java Card specialties. The main actors are marked with bold letters.

Phase	Name	Description
1	Smartcard Embedded Software Development	The **Smartcard Embedded Software Developer** is in charge of • smartcard embedded software development including the development of Java applets and • specification of IC pre-personalization requirements, though the actual data for IC pre-personalization come from phase 6 (or phase 4 or 5).

Phase	Name	Description
2	IC Development	The **IC Designer** • designs the IC, • develops IC Dedicated Software, • provides information, software or tools to the Smartcard Embedded Software Developer, and • receives the smartcard embedded software from the developer, through trusted delivery and verification procedures. From the IC design, IC Dedicated Software and Smartcard Embedded Software, the **IC Designer** • constructs the smartcard IC database, necessary for the IC photomask fabrication.
3	IC Manufacturing and Testing	The **IC Manufacturer** is responsible for • producing the IC through three main steps: IC manufacturing, IC testing, and IC pre-personalization. The IC Mask Manufacturer • generates the masks for the IC manufacturing based upon an output from the smartcard IC database.
4	IC Packaging and Testing	The **IC Packaging Manufacturer** is responsible for • IC packaging and testing.
5	Smartcard Product Finishing Process	The **Smartcard Product Manufacturer** is responsible for • smartcard product finishing process including applet loading and testing.
6	Smartcard Personalization	The **Personalizer** is responsible for • smartcard (including applet) personalization and final tests. Other smartcard embedded software may be loaded onto the chip at the personalization process,
7	Smartcard End-usage	The **Smartcard Issuer** is responsible for • smartcard product delivery to the smartcard end-user, and the end of life process.

The evaluation process is limited to phases 1 to 4, while delivery is either in phase 3 or 4 (see also [ST0348]). The delivery comprises the following items:

- TOE: Java Card NXP P541G072V0P (JCOP 41 v2.3.1),
- Administrator guidance for smart card finishing and personalizing including applet loading (phases 3, 5, 6) [AGD_ADM]
- User guidance for the applet developer (phase 1) [AGD_USR]

Applet development is outside the scope of this evaluation. Applets with patch code can be loaded in phase 3 only. Normal applet loading is only possible in phases 5 or 6, i. e. no post-issuance loading of applets.

Following [PP0002] phase 2 “IC Development” and phase 3 “IC Manufacturing and Testing” are under the responsibility of the IC manufacturer, i. e. the role IC manufacturer includes from here on also the role IC designer. (see also section 3.2)

2.3 Java Card Technology

This section is partly taken from [JCSPP] and should be considered as an introduction to Java Card Technology:

Java Card technology combines a subset of the Java programming language with a runtime environment optimized for smart cards and similar small-memory embedded devices [JCVM21]. The Java Card platform is a smart card platform enabled with Java Card technology (also called a “Java card”). This technology allows for multiple applications to run on a single card and provides facilities for secure interoperability of applications. Applications for the Java Card platform (“Java Card applications”) are called *applets*.

The version 2.2.1 of the Java Card platform is specified in [JCVM221], [JCRE221] and [JCAPI221]. It consists of the virtual machine for the Java Card platform (“Java Card virtual machine” or “*JCVM*”), the Java Card technology runtime environment (*JCRE*) and the Java Card Application Programming Interface (API).

As the terminology is sometimes confusing, we introduce the term “Java Card System” to designate the set made of the *JCRE*, the *JCVM* and the API. The *Java Card System* provides a layer between a **native platform** and an *applet* space. That layer allows applications written for one smart card platform (“*SCP*“) enabled with Java Card technology to run on any other such platform.

The ***JCVM*** is essentially an abstract machine that specifies the behavior of the bytecode interpreter to be embedded in the card. The ***JCRE*** is responsible for card resource management, communication, *applet* execution, and on-card system and *applet* security. The API provides classes and interfaces for the core functionality of a Java Card application. It defines the calling conventions by which an *applet* may access the *JCRE* and native services such as, I/O management functions, PIN and cryptographic specific management and the exceptions mechanism. The Java Card API is compatible with formal international standards, such as ISO7816, and industry specific standards, such as EMV (Europay/Master Card/Visa).

In certain use-cases, *applet*s can be loaded and installed on a Java Card platform after the card has been issued[2]. This provides, for instance, card issuers with the ability to dynamically respond to their customer’s changing needs. For example, if a customer decides to change the frequent flyer program associated with the card, the card issuer can

[2] For the present TOE, there is no post-issuance loading of applets. (see also section 2.2)

make this change, without having to issue a new card. Moreover, *applet*s from different vendors can coexist in a single card, and they can even share information. An *applet*, however, is usually intended to store highly sensitive information, so the sharing of that information must be carefully limited. In the Java Card platform *applet* isolation is achieved through the *applet firewall* mechanism ([JCRE221], §6.1). That mechanism confines an *applet* to its own designated memory area, thus each *applet* is prevented from accessing fields and operations of objects owned by other *applet*s, unless an interface is explicitly provided (by the *applet* who owns it) for allowing access to that information. The *firewall* is dynamically enforced, that is, at runtime by the *JCVM.* However *applet* isolation cannot entirely be granted by the *firewall* mechanism if certain integrity conditions are not satisfied by the applications loaded on the card. Those conditions can be statically verified to hold by a bytecode verifier.

The development of the source code of the *applet* is carried on in a Java programming environment. The compilation of that code will then produce the corresponding *class* file. Then, this latter file is processed by the **converter**[3], which, on the one hand, validates the code and generates a converted applet (*CAP*) file, the equivalent of a Java™ class file for the Java Card platform. A *CAP file* contains an executable binary representation of the classes of a *package*. A *package* is a name space within the Java programming language that may contain *class*es and *interface*s, and in the context of Java Card technology, it defines either a user library, or one or several *applets.* Then, the integrity of the *CAP file* is checked by the (off-card) **bytecode verifier**. After the validation is carried out, the *CAP file* is then loaded into the card making use of a safe **loading** mechanism. Once loaded into the card the file is **linked**, what makes it possible in turn to **install**, if defined, instances of any of the *applet*s defined in the file. During the installation process the *applet* is registered on the card by using an application identifier (*AID*). This *AID* will allow the identification of unique instances of the *applet* instance within the card. In particular, the AID is used for selecting the applet instance for execution. The execution of the *applet*'s code is performed by the bytecode interpreter residing on the card.

The following sections further describe some of the components involved in the environment of the *Java Card System.* Although most of those components are not part of the TOE, a better understanding of the role they play will help in understanding the importance of the assumptions that will appear concerning the environment of the TOE.

A brief description of some of the new features introduced in the version 2.2.1 of the Java Card platform is also included.

2.3.1 Bytecode Verification

The bytecode verifier is a program that performs static checks on the bytecodes of the methods of a *CAP file.* Bytecode verification is a ***key*** component of security: applet isolation, for instance, depends on the file satisfying the properties a verifier checks to hold. A method of a *CAP file* that has been verified, shall not contain, for instance, an instruction

[3] The converter is defined in the specifications so as to being the off-card component of the *JCVM.*

that allows forging a memory address or an instruction that makes improper use of a return address as if it were an object reference. In other words, bytecodes are verified to hold up to the intended use to which they are defined. This document considers static bytecode verification, which is performed on the host (*off-card* verification) and prior to the installation of the file on the card in any case. However, part of the verifications on bytecodes might be performed totally or partially dynamically. No standard procedure in that concern has yet been recognized. Furthermore, different approaches have been proposed for the implementation of bytecode verifiers, most notably data flow analysis, model checking and lightweight bytecode verification, this latter being an instance of what is known as proof carrying code. The actual set of checks performed by the verifier is implementation-dependent, but it is required that it should at least enforce all the "must clauses" imposed in [JCVM] on the bytecodes and the correctness of the *CAP file*s' format.

2.3.2 Installation of Applets

The *installer* is the part of the on-card component of the platform dealing with downloading, linking and installation of new *package*s, as described in [JCRE221]. Once selected, it receives the *CAP file*, stores the classes of the *package* on the card, initializes static data, if any, and installs any applets contained in the package.

In some cases, the actual installation (and registration) of *applet*s is postponed; in the same vein, a *package* may contain several *applet*s, and some of them might never be installed. Installation is then usually separated from the process of loading and linking a *CAP file* on the card.

When post-issuance installation of *applets* is supported by a Java Card platform[4], processes that allow to *load*, and also to *link*, a *CAP file*, as well as to install *applet* instances on the card, must also be provided. If post-issuance installation is supported then the *installer* is also considered as part of the *Java Card System*.

2.3.2.1 Loading

The loading of a file into the card embodies two main steps: First an authentication step by which the card issuer and the card recognize each other, for instance by using a type of cryptographic certification. Once the identification step is accomplished, the *CAP file* is transmitted to the card by some means, which in principle should not be assumed to be secure. Due to resource limitations, usually the file is split by the card issuer into a list of Application Protocol Data Units (*APDU*s), which are in turn sent to the card.

2.3.2.2 Linking

The linking process consists of a rearrangement of the information contained in the *CAP file* in order to speed up the execution of the applications. There is a first step where indirect external and internal references contained in the file are resolved, by replacing those references with direct ones. This is what is referred to as the *resolution* step. In the next

[4] For the present TOE, there is no post-issuance loading of applets. (see also section 2.2)

step, called in [JVM] the *preparation* step, the static field image[5] and the statically initialized arrays defined in the file are allocated. Those arrays in turn are also initialized, thus giving rise to what shall constitute the initial state of the *package* for the embedded interpreter.

2.3.3 The Card Manager (CM)

The card manager is an application with specific rights, which is responsible for the administration of the smart card. This component will in practice be tightly connected with the *JCRE*. The card manager is in charge of the life cycle of the whole card, as well as the installed applications (*applet*s). It may have other roles (such as the management of security domains and enforcement of the card issuer security policies) that we do not detail here, as they are not in the scope of the TOE and are implementation–dependent.

The card manager's role is also to manage and control the communication between the card and the card acceptance device (*CAD*). For this ST, the card manager belongs to the card.

2.3.4 Smart Card Platform

The smart card platform (*SCP*) is composed of a micro-controller and hardware abstraction layer (see section 2.1). No separate operating system is present in this card. It provides memory management functions (such as separate interface to RAM and NVRAM), I/O functions that are compliant with ISO standards, transaction facilities, and secure implementation of cryptographic functions.

The hardware platform, the Philips NXP P5CT072V0P Secure Smart Card Controller, has been certified at BSI under registration number BSI-DSZ-CC-0348-2006 to EAL5 augmented by ALC_DVS.2, AVA_MSU.3, and AVA_VLA.4.

The present evaluation is a composite evaluation using the results of this hardware certification.

2.3.5 Native Applications

Apart from Java Card applications, the final product may contain native applications as well. Native applications are outside the scope of the TOE security functions (TSF), and they are usually written in the assembly language of the platform, hence their name. This term also designates software libraries providing services to other applications, *including applet*s under the control of the TOE.

It is obvious that such native code presents a threat to the security of the TOE and to user *applets*.

Therefore, [JCSPP] will require for native applications to be conformant with the TOE so as to ensure that they do not provide a means to circumvent or jeopardize the TSFs.

[5] The memory area where the static fields of the file reside.

For the present product, the certified hardware contains a native MIFARE application that belongs to the TOE. A TOE configured with the minor configuration option “MIFARE Emulation = A” does not provide an additional interface to the environment because the MIFARE application is logically disabled. In the minor configurations “MIFARE Emulation = B1” and “MIFARE Emulation = B4” the contactless MIFARE Classic OS is implemented and has access to 1KB and 4KB EEPROM memory respectively. (see section 2.1) The final product does not contain any other native applications according to [JCSPP]. To completely securely separate the User OS and the MIFARE OS, which is enabled in minor configurations “MIFARE Emulation = B1” and “MIFARE Emulation = B4”, the hardware provides the CVEC firewall.

2.4 Java Functional Components

In section 1.7.1 of [JCSPP] the concept of group of SFRs was introduced and the role they play in the PPs is defined in this document. The following list describes the groups of security requirements which have been used in those PPs:

SCP group	The *SCPG* contains the security requirements for the smart card platform, that is, operating system and chip that the Java Card System is implemented upon. In the present case, this group applies to the TOE and is within the scope of evaluation.
Core group	The *CoreG* contains the basic requirements concerning the runtime environment of the Java Card System, such as the firewall policy and the requirements related to the Java Card API. This group is within the scope of evaluation.
Bytecode verification group	The *BCVG* contains the security requirements concerning the bytecode verification of the application code to be loaded on the card. In the present case, this group of SFRs applies to the IT environment.
Installation group	The *InstG* contains the security requirements concerning the installation of post-issuance applications. It does not address card management issues in the broad sense, but only those security aspects of the installation procedure that are related to applet execution. Those aspects are described in §11.1.5 *Installer behavior* of [JCRE221]. This group is not within the scope of evaluation.
Applet deletion group	The *ADELG* contains the security requirements for erasing installed applets from the card, a new feature introduced in Java Card System 2.2.2. It can also be used as a basis for any other application deletion requirements. This group is not within the scope of evaluation.

Remote Method Invocation (RMI) group	The *RMIG* contains the security requirements for the remote method invocation features, which provides a new protocol of communication between the terminal and the applets. This was introduced in Java Card System 2.2.2. This group is not within the scope of evaluation.
Logical channels group	The *LCG* contains the security requirements for the logical channels, which provide a runtime environment where several applets can be simultaneously selected or a single one can be selected more than once. This is a Java Card System 2.2.2 feature. This group is not within the scope of evaluation.
Object deletion group	The *ODELG* contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card System 2.2.2 feature. This group is not within the scope of evaluation.
Secure carrier group	The *CarG* group contains minimal requirements for secure downloading of applications on the card. This group contains the security requirements for preventing, in those configurations which do not support on-card static or dynamic verification of bytecodes, the installation of a *package* that has not been bytecode verified, or that has been modified after bytecode verification. This group is not within the scope of evaluation.
Card manager group	The *CMGRG* contains the minimal requirements that allow defining a policy for controlling access to card content management operations and for expressing card issuer security concerns. This group is within the scope of evaluation.

The same concept of groups is also used in this security target.

2.4.1 Scope of Evaluation

The following groups of SFRs defined in [JCSPP] are within the scope of this Security Target:

- Core (CoreG) (part of the TOE)
- Bytecode verification (BCVG) (part of the IT environment)
- Smart card platform (SCPG) (part of the TOE)
- Card manager (CMGRG) (part of the TOE)

Let us also remark that the code of the applets is not part of the code of the TOE, but just data managed by the TOE. Moreover, the scope of the ST does not include all the stages in the development cycle of a Java Card application described in section 2.2. Applets are only considered in their CAP format, and the process of compiling the source code of an application and converting it into the CAP format does not regard the TOE or its environment. On the contrary, the process of verifying applications in its CAP format and loading it on the card is a crucial part of the TOE environment and plays an important role

as a complement of the TSFs. The ST assumes that the loading of applications pre-issuance is made in a secure environment.

2.5 TOE Intended Usage

Smart cards are mainly used as data carriers that are secure against forgery and tampering. More recent uses also propose them as personal, highly reliable, small size devices capable of replacing paper transactions by electronic data processing. Data processing is performed by a piece of software embedded in the smart card chip, usually called an application.

The *Java Card System* is intended to transform a smart card into a platform capable of executing applications written in a subset of the Java programming language. The intended use of a Java Card platform is to provide a framework for implementing IC independent applications conceived to safely coexist and interact with other applications into a single smart card.

Applications installed on a Java Card platform can be selected for execution when the card is inserted into a card reader. In some configurations of the TOE, the card reader may also be used to enlarge or restrict the set of applications that can be executed on the Java Card platform according to a well-defined card management policy.

Notice that these applications may contain other confidentiality (or integrity) sensitive data than usual cryptographic keys and PINs; for instance, passwords or pass-phrases are as confidential as the PIN, and the balance of an electronic purse is highly sensitive with regard to arbitrary modification (because it represents real money).

So far, the most important applications are:

- Financial applications, like Credit/Debit ones, stored value purse, or electronic commerce, among others.
- Transport and ticketing, granting pre-paid access to a transport system like the metro and bus lines of a city.
- Telephony, through the subscriber identification module (SIM) for digital mobile telephones.
- Personal identification, for granting access to secured sites or providing identification credentials to participants of an event.
- Electronic passports and identity cards.
- Secure information storage, like health records, or health insurance cards.
- Loyalty programs, like the “Frequent Flyer” points awarded by airlines. Points are added and deleted from the card memory in accordance with program rules. The total value of these points may be quite high and they must be protected against improper alteration in the same way that currency value is protected.

3 TOE Security Environment

This chapter describes the security aspects of the environment in which the TOE is intended to be used and addresses assets, threats, organizational security policies, and assumptions. The last section describes some general security aspects that are relevant for the Java Card functionality. It is intended to ease the comprehension of the security objectives and requirements, especially the access control policies.

The description is based on [PP0002] and supplemented by the description of [JCSPP].

3.1 Assets

In this section assets are divided in primary and secondary assets. The primary assets User Data and TSF Data are further refined, and the definition of section 3.2 of [JCSPP] is taken.

The TOE objective is to protect assets, the primary assets, during usage phase. In order to protect these primary assets, information and tools used for the development and manufacturing of the Smart Card, need to be protected. These information and tools are called secondary assets.

Assets have to be protected, some in terms of confidentiality and some in terms of integrity or both integrity and confidentiality.

- Primary assets (to be protected by the TOE):

  These assets are concerned by the threats on the TOE and include:

  - TOE including NOS[6] code,
  - TSF data, as initialization data, configuration data, cryptographic keys, random numbers for key generation, and all data used by the TOE to execute its security functions. This includes also configuration of hardware specific security features.
  - User Data, as application code (applets), specific sensitive application values, as well as application specific PIN and authentication data.

- Secondary assets (to be protected by the environment):

  These are the information, data and tools used for the development and manufacturing of the TOE.

  - IC development and manufacturing related information, handled by the IC manufacturer during phase 2 and 3 as IC specification, IC dedicated software. These assets are concerned by the T.DEV_IC threat.
  - NOS development related information handled by NOS developer during phase 1. These assets are concerned by the T.DEV_NOS threat

[6] Remark: The expression native operating system (NOS) is defined as the smart card embedded software consisting of runtime environment (RTE), virtual machine (VM), Java Card API, and card manager (CM). (see figure 1 in section 2.1).

- TOE documentation exchanged between IC manufacturer and NOS developer as IC data sheet, IC user guidance, NOS mask related information. These assets are concerned by the T.DEL_IC_NOS threat.
- TOE documentation delivered to IC packaging or Smartcard product manufacturer as initialization data or other sensitive information for usage phase 4 to 7. These assets are concerned by the T.DEL threat.

The relation of these threats for the environment to the life-cycle phases and assets is illustrated in the following figure:

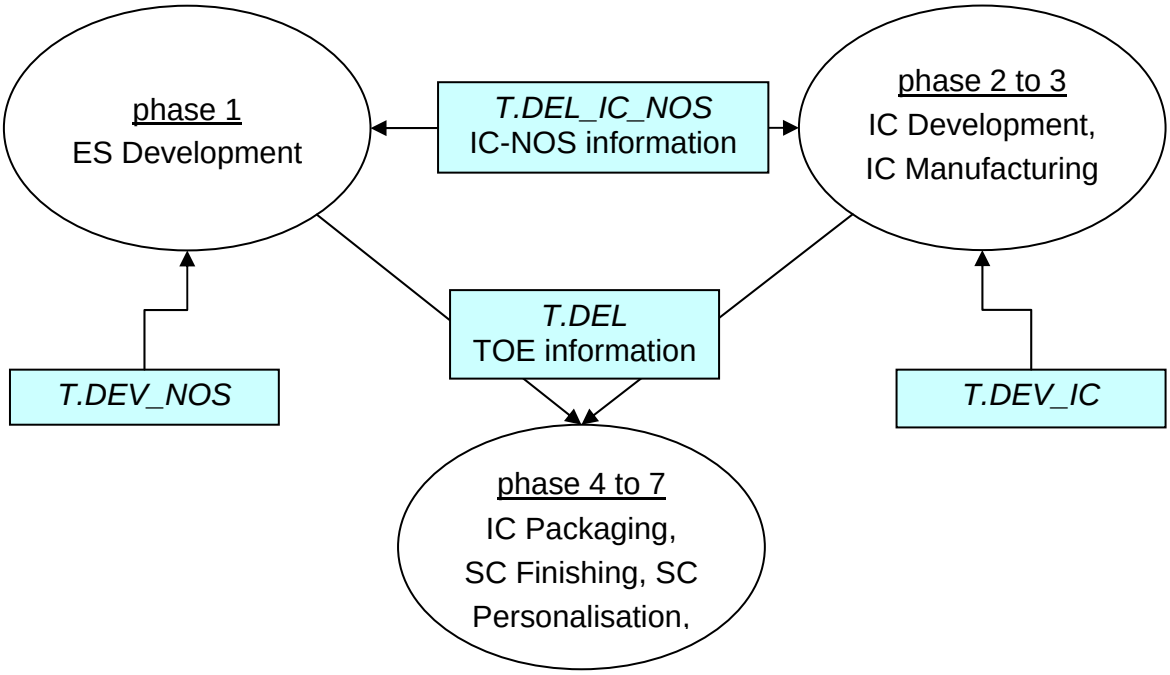


Figure 2: Threats in the TOE environment

Java Card specific (primary) assets defined in [JCSPP] to be protected by the TOE are listed below. They are grouped according to whether it is data created by and for the user (User data) or data created by and for the TOE (TSF data). For each asset it is specified the kind of dangers that weighs on it.

3.1.1 User Data

D.APP_CODE The code of the *applet*s and libraries loaded on the card.

To be protected from unauthorized modification.

D.APP_C_DATA Confidential sensitive data of the applications, like the data contained in an object, a static field of a *package*, a local variable of the currently executed method, or a position of the operand stack.

To be protected from unauthorized disclosure.

D.APP_I_DATA Integrity sensitive data of the applications, like the data contained in an object, a static field of a *package*, a local variable of the currently executed method, or a position of the operand stack.

To be protected from unauthorized modification.

D.PIN Any end-user's PIN.

To be protected from unauthorized disclosure and modification.

D.APP_KEYs Cryptographic keys owned by the *applet*s.

To be protected from unauthorized disclosure and modification.

3.1.2 TSF Data

D.JCS_CODE The code of the *Java Card System*.

To be protected from unauthorized disclosure and modification.

D.JCS_DATA The internal runtime data areas necessary for the execution of the *JCVM*, such as, for instance, the frame stack, the program counter, the class of an object, the length allocated for an array, any pointer used to chain data-structures.

To be protected from monopolization and unauthorized disclosure or modification.

D.SEC_DATA The runtime security data of the *JCRE*, like, for instance, the *AID*s used to identify the installed *applet*s, the *Currently selected applet*, the *current context* of execution and the owner of each object.

To be protected from unauthorized disclosure and modification.

D.API_DATA Private data of the API, like the contents of its private fields

To be protected from unauthorized disclosure and modification.

D.JCS_KEYs Cryptographic keys used when loading a file into the card.

To be protected from unauthorized disclosure and modification.

D.CRYPTO Cryptographic data used in runtime cryptographic computations, like a seed used to generate a key.

To be protected from unauthorized disclosure and modification.

3.2 Users & Subjects

The life-cycle contains the following actors: (see also section 2.2)

- **NOS Developer:** includes the roles NOS developer and application developer (phase 1). In the present case the NOS is JCOP 41 and the developer is IBM.
- **IC Manufacturer** includes the roles IC designer (phase 2) and IC manufacturer (phase 3). In the present case the IC is the NXP P5CT072V0P Secure Smart Card Controller and the IC manufacturer is Philips.
- **Card Manufacturer** includes the roles IC Packaging Manufacturer and Smartcard Product Manufacturer and is responsible for IC Packaging and finishing process (phases 4-5)

The following description is taken over from Protection profile [JCSPP]:

*Subject*s are active components of the TOE that (essentially) act on the behalf of *users*. The users of the TOE include people or institutions (like the applet developer, the card issuer, the verification authority), hardware (like the *CAD* where the card is inserted) and software components (like the application *package*s installed on the card). Some of the users may just be aliases for other users. For instance, the verification authority in charge of the bytecode verification of the applications may be just an alias for the card issuer or in the case of this ST, the IC manufacturer.

The main *subject*s of the TOE considered in this document are the following ones:

- *Package*s used on the Java Card platform that act on behalf of the applet developer. These subjects are involved in the ***FIREWALL security policy*** defined in §5.1.1 and they should be understood as instances of the subject *S.PACKAGE*.

- The *CardManager,* can be considered a special instance of S.PACKAGE which implements the Open Platform specification. This package provides the functionality of a runtime environment running at the JCRE 'system' (privileged) context and for clarity is always represented by the subject *S.PACKAGE(CM)*.

- The *JCRE*, which acts on behalf of the card issuer. This subject is involved in several of the security policies defined in this document and is always represented by the subject *S.JCRE*.

Note: The subjects from [JCSPP]:

- (on-card) *bytecode verifier*
- *installer*
- *applet deletion manager*.

are not relevant for the minimal configuration and for this ST.

3.3 Assumptions

This section introduces the assumptions made on the environment of the TOE. They are summarized in the following table together with the life-cycle phase they apply to:

Name	Source	Refined?	Life-Cycle
A.DLV_PROTECT	-	-	phases 4-7
A.TEST_OPERATE	-	-	phases 4-6
A.USE_DIAG	-	-	phase 7
A.USE_KEYS	-	-	phase 7
A.NATIVE	[JCSPP]	no	phases 1-6
A.NO-DELETION	[JCSPP]	no	phase 7
A.NO-INSTALL	[JCSPP]	no	phase 7
A.VERIFICATION	[JCSPP]	no	phases 1-6

Table 1: Assumptions

3.3.1 Assumptions not contained in [JCSPP]

3.3.1.1 Assumptions on the TOE Delivery Process (Phases 4 to 7)

Procedures shall guarantee the control of the TOE delivery and storage process and conformance to its objectives as described in the following assumptions:

A.DLV_PROTECT Procedures shall ensure protection of TOE material/information under delivery and storage.

Procedures shall ensure that corrective actions are taken in case of improper operation in the delivery process and storage.

Procedures shall ensure that people dealing with the procedure for delivery have got the required skill.

3.3.1.2 Assumptions on Phases 4 to 6

A.TEST_OPERATE It is assumed that security procedures are used during all manufacturing and test operations through phases 4, 5, 6 to maintain confidentiality and integrity of the TOE and of its manufacturing and test data (to prevent any possible copy, modification, retention, theft or unauthorized use).

It is assumed that appropriate functionality testing of the TOE is used in phases 4, 5 and 6.

3.3.1.3 Assumption on Phase 7

A.USE_DIAG It is assumed that the environment supports and uses the secure communication protocols offered by TOE.

A.USE_KEYS	It is assumed that the keys which are stored outside the TOE and which are used for secure communication and authentication between Smart Card and terminals are protected for confidentiality and integrity in their own storage environment. Application note: This is to assume that the keys used in terminals or systems are correctly protected for confidentiality and integrity in their own environment, as the disclosure of such information which is shared with the TOE but is not under the TOE control, may compromise the security of the TOE.

3.3.2 Assumptions from [JCSPP]

A.NATIVE	Those parts of the APIs written in native code as well as any pre-issuance native application on the card are assumed to be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p. 36) for details
Remark:	A.NATIVE is related to all phases except phase 7 as in the usage phase, no more native code can be loaded onto the card
A.NO-DELETION	No deletion of installed *applet*s (or *package*s) is possible.
Remark:	A.NO-DELETION is related to phase 7.
A.NO-INSTALL	There is no post-issuance installation of *applet*s. Installation of *applet*s is secure and occurs only in a controlled environment in the pre-issuance phase. See #.*INSTALL* (p.38) for details.
Remark:	This assumption is related to phase 7.
A.VERIFICATION	All the bytecodes are verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time.
Remark:	A.VERIFICATION is related to phases 1-6 since in the present case, bytecode verification is performed before loading.

3.4 Threats

This section introduces the threats to the assets against which specific protection within the TOE or its environment is required. It is assumed that all attackers have high level of expertise, opportunity and resources. General threats for smart card native operating systems were defined and supplemented by Java Card specific threats from [JCSPP]. They are summarized in the following table together with the life-cycle phase they apply to:

Name	Source	Refined?	Life-Cycle
T.DEV_IC	-	-	phases 2-3
T.DEV_NOS	-	-	phase 1
T.DEL_IC_NOS	-	-	phases 1-2
T.DEL	-	-	phases 4-6
T.ACCESS_DATA	-	-	phase 7
T.OS_OPERATE	-	-	phase 7
T.OS_DECEIVE	-	-	phase 7
T.LEAKAGE	-	-	phase 7
T.FAULT	-	-	phase 7
T.RND	[PP0002]	no	phase 7
T.PHYSICAL	[JCSPP]	yes[7]	phase 7
T.CONFID-JCS-CODE	[JCSPP]	no	phase 7
T.CONFID-APPLI-DATA	[JCSPP]	no	phase 7
T.CONFID-JCS-DATA	[JCSPP]	no	phase 7
T.INTEG-APPLI-CODE	[JCSPP]	no	phase 7
T.INTEG-JCS-CODE	[JCSPP]	no	phase 7
T.INTEG-APPLI-DATA	[JCSPP]	no	phase 7
T.INTEG-JCS-DATA	[JCSPP]	no	phase 7
T.SID.1	[JCSPP]	no	phase 7
T.SID.2	[JCSPP]	no	phase 7
T.EXE-CODE.1	[JCSPP]	no	phase 7
T.EXE-CODE.2	[JCSPP]	no	phase 7
T.NATIVE	[JCSPP]	no	phase 7
T.RESOURCES	[JCSPP]	no	phase 7

Table 2: Threats

[7] Refinement to cover additional aspects of O.SCP.IC not contained in [JCSPP].

3.4.1 Threats not contained in [JCSPP]

The TOE is required to counter the threats described hereafter; a threat agent wishes to abuse the assets either by functional attacks or by environmental manipulation, by specific hardware manipulation, by a combination of hardware and software manipulations or by any other type of attacks.

Threats have to be split in

- Threats against which specific protection within the TOE is required,
- Threats against which specific protection within the environment is required.

3.4.1.1 Unauthorized full or partial Cloning of the TOE

The cloning of the functional behavior of the Smart Card on its ISO command interface is the highest-level security concern in the application context. The cloning of that functional behavior requires:

- To develop a functional equivalent of the Smart Card Native Operating System and its applications, to disclose, to interpret and employ the secret User Data stored in the TOE, and
- To develop and build a functional equivalent of the Smart Card using the input from the previous steps.

The Native Operating System must ensure that especially the critical User Data are stored and processed in a secure way but also ensures that critical User Data are treated as required in the application context. In addition, the personalization process supported by the Smart Card Native Operating System (and by the Smart Card Integrated Circuit in addition) must be secure.

This last step is beyond the scope of this Security Target. As a result, the threat “cloning of the functional behavior of the Smart Card on its ISO command interface” is averted by the combination of measures, which split into those being evaluated according to this Security Target and the corresponding personalization process. Therefore, functional cloning is indirectly covered by the threats described below.

3.4.1.2 Threats on TOE Environment

Note: Threats on TOE environment have been regrouped in threats during NOS development, threats on IC during development & manufacturing, threats on IC and NOS related information during delivery, threat on TOE information during delivery to users (phase 4 to 7).

One threat addresses one environment, thus regrouping disclosure, theft, and modification.

T.DEV_IC	Theft, modification, disclosure of information related to IC development and manufacturing. This includes disclosure/modification of the NOS code by the IC manufacturer.

	An attacker may gain access to IC development and manufacturing information and be able to gain knowledge on the IC and the IC security mechanism implementation, or steal sensitive information that could be used for future attacks or cloning purpose.
Complementary note	This threat addresses the information handled by the IC manufacturer in the IC development and manufacturing environment (phases 2 and 3).
T.DEV_NOS	Theft, modification, or disclosure of NOS related information during NOS development. An attacker may gain access to NOS development information, and be able to: o Modify the implementation code or data (e.g. introducing a trap door in NOS), o Steal or use development tools, test program or samples (allowing to fake TOE behavior) o Gain knowledge of NOS specification, implementation of security mechanism , including IC information delivered by the IC manufacturer, enabling further attacks in usage phase.
Complementary note	This threat addresses the information handled by the NOS Developer during phase 1.
T.DEL_IC_NOS	Theft, modification, disclosure of information related to IC or NOS during delivery between IC manufacturer and NOS Developer. An attacker may get information on TOE implementation during the delivery operation between IC manufacturer and NOS developer. The knowledge or the modification of the transferred information could allow future attacks in usage phase. The information exchanged between NOS developer and IC developer includes the ROM mask, pre-initialization data, IC documentation delivered by IC manufacturer.
Complementary note	This threat addresses the delivery process used for information exchange between the IC manufacturer and the NOS developer.
T.DEL	Theft, modification, disclosure of information related to TOE during delivery to IC packaging manufacturer or Smart Card manufacturer or personalize.

	An attacker may get information on TOE application or configuration, during delivery of TOE sensitive data to IC packaging Smart Card Manufacturer or personalizer. The knowledge or the modification of the transferred information could allow future attacks in usage phase.
Complementary note	This threat addresses the delivery process used for information transfer to IC packaging, Smart Card Manufacturer, or Personalizer.

3.4.1.3 Threats on Phases 4 to 7

The TOE is intended to protect itself against the following threats

- Manipulation of User Data and of the Smart Card Native Operating System (while being executed/processed and while being stored in the TOE's memories) and
- Disclosure of User Data and of the Smart Card NOS (while being processed and while being stored in the TOE's memories).

The above threats are derived from considering the end-usage phase (phase 7) since phases 1, phases 2 and 3 (IC development and production) and TOE Delivery up to the beginning of phase 4 are covered by organizational aspects.

The phases 4 to 6 are clearly out of the scope of this Security Target and the related organizational aspects are not discussed in this section.

The TOE's countermeasures are designed to avert the threats described below. Nevertheless, they may be effective in earlier phases (phases 4 to 6).

Though the Native Operating System (normally stored in the ROM) will in many cases not contain secret data or algorithms, it must be protected from being disclosed, since for instance knowledge of specific implementation details may assist an attacker. In many cases critical User Data and NOS configuration data (TSF data) will be stored in the EEPROM.

3.4.1.3.1 Software Threats

The most basic function of the Native Operating System is to provide data storage and retrieval functions with a variety of access control mechanisms which can be configured to suit the embedded application(s) context requirements.

Each authorized role has certain specified privileges which allow access only to selected portions of the TOE and the information it contains. Access beyond those specified privileges could result in exposure of assets. On another hand, an attacker may gain access to sensitive data without having permission from the entity that owns or is responsible for the information or resources.

T.ACCESS_DATA	Unauthorized access to sensitive information stored in memories in order to disclose or to corrupt the TOE data (TSF and user data).

This includes any consequences of bad or incorrect user authentication by the TOE.

Several software attack methods may be used here, as:

- Brute force data space search attacks,
- Administrator or user authentication failures information
- Monitoring TOE inputs/outputs,
- Replay attacks,
- Cryptographic attacks,

Regarding direct attacks on the Native Operating System, there are series of attack paths that address logical probing of the TOE. These are brute force data search, replay attack, insertion of faults, and invalid input.

T.OS_OPERATE Modification of the correct NOS behavior by unauthorized use of TOE or use of incorrect or unauthorized instructions or commands or sequence of commands, in order to obtain an unauthorized execution of the TOE code.

An attacker may cause a malfunction of TSF or of the Smart Card embedded NOS in order to (1) bypass the security mechanisms (i.e. authentication or access control mechanisms) or (2) obtain unexpected result from the embedded NOS behavior

Different kind of attack path may be used as:

- Applying incorrect unexpected or unauthorized instructions, commands or command sequences,
- Provoking insecure state by insertion of interrupt (reset), premature termination of transaction or communication between IC and the reading device

Complementary note Any implementation flaw in the NOS itself can be exploited with this attack path to lead to an unsecured state of the state machine of the NOS.

The attacker uses the available interfaces of the TOE.

A user could have certain specified privileges that allow loading of selected programs. Unauthorized programs, if allowed to be loaded, may include either the execution of legitimate programs not intended for use during normal operation (such as patches, filters, Trojan horses, etc.) or the unauthorized loading of programs specifically targeted at penetration or modification of the security functions. Attempts to generate a non-secure state in the Smart Card may also be made through premature termination of transactions or communications between the IC and the card reading device, by insertion of interrupts, or by selecting related applications that may leave files open.

T.OS_DECEIVE Modification of the expected TOE configuration by

- unauthorized loading of code,

- unauthorized execution of code
- unauthorized modification of code behavior

Complementary note Any software manipulations on TOE application data (User or TSF data) by interaction with other program or security features that may not be used after one given life cycle state can lead to unsecured state.

The attacker needs to know specific information about the TOE implementation. Loading of code in EEPROM (patch or filter) is considered here as authorized, if this code is part of the evaluation and if loading operation is performed by an authorized administrator in the defined environment.

3.4.1.3.2 Environment Threats on the complete TOE

Regarding physical point of view, the TOE is exposed to different types of influences or interactions with its outer world. One can take advantage of the leakage from the TOE to disclose assets or derived data.

T.LEAKAGE An attacker may exploit information which is leaked from the TOE during usage of the Smart Card in order to disclose the confidential primary assets.

This attack is non-invasive and requires no direct physical contact with the Smart Card Internals. Leakage may occur through emanations, variations in power consumption, I/O characteristics, clock frequency, or by changes in processing time requirements. One example is the Differential Power Analysis (DPA).

Another security concern is to take advantage of the susceptibility of the integrated circuit to put the TOE in an unsecured state.

T.FAULT An attacker may cause a malfunction of TSF or of the Smart Card embedded NOS by applying environmental stress in order to (1) deactivate or modify security features or functions of the TOE or (2) deactivate or modify security functions of the Smart Card embedded NOS. This may be achieved by operating the Smart Card outside the normal operating conditions

Complementary note A composition attack paths can take advantage of any susceptibility of the whole product (ES and IC):

- environmental variations (temperature, power supply, clock frequency) or exposition to strenuous particles (light and electrical waves)

The attacker uses the available interfaces of the TOE

3.4.1.3.3 Threat on Random Numbers

The following threat was taken over from [PP0002]:

T.RND — Deficiency of Random Numbers

An attacker may predict or obtain information about random numbers generated by the TOE for instance because of a lack of entropy of the random numbers provided.

An attacker may gather information about the produced random numbers which might be a problem because they may be used for instance to generate cryptographic keys.

Here the attacker is expected to take advantage of statistical properties of the random numbers generated by the TOE without specific knowledge about the TOE's generator. Malfunctions or premature ageing are also considered which may assist in getting information about random numbers.

3.4.2 Threats from [JCSPP]

The following threats specific for the Java Card functionality were taken from [JCSPP].

T.PHYSICAL — The attacker **discloses** or **modifies** the design of the TOE, its sensitive data (TSF and User Data) or application code or **disables** security features of the TOE using pure invasive, physical (opposed to logical) attacks on the hardware part of the TOE.

These attacks are performed using physical probing or physical manipulation of the hardware (reverse engineering, manipulation of memory cells, manipulation of hardware security parts) and include IC failure analysis, electrical probing, unexpected tearing, and DP analysis. That also includes the modification of the runtime execution of *Java Card System* or *SCP* software through alteration of the intended execution order of (set of) instructions through physical tampering techniques.

This threatens *all* the identified assets.

This threat refers to *#.SCP.7*, and all aspects related to confidentiality and integrity of code and data.

Note: This threat from [JCSPP] was refined to cover additional aspects of O.SCP.IC not contained in [JCSPP].

3.4.2.1 Confidentiality

T.CONFID-JCS-CODE — The attacker executes an application without authorization to disclose the *Java Card System* code. See #.*CONFID-JCS-CODE* (p. 35) for details.

Directly threatened asset(s): **D.JCS_CODE**.

T.CONFID-APPLI-DATA The attacker executes an application without authorization to disclose data belonging to another application. See #.*CONFID-APPLI-DATA* (p. 35) for details.

Directly threatened asset(s): **D.APP_C_DATA**, **D.PIN** and **D.APP_KEYs**.

T.CONFID-JCS-DATA The attacker executes an application without authorization to disclose data belonging to the *Java Card System*. See #.*CONFID-JCS-DATA* (p. 35) for details.

Directly threatened asset(s): **D.API_DATA**, **D.SEC_DATA**, **D.JCS_DATA** **D.JCS_KEYs** and **D.CRYPTO**.

3.4.2.2 Integrity

T.INTEG-APPLI-CODE The attacker executes an application to alter (part of) its own or another application's code. See #.*INTEG-APPLI-CODE* (p. *35*) for details.

Directly threatened asset(s): **D.APP_CODE**

T.INTEG-JCS-CODE The attacker executes an application to alter (part of) the *Java Card System* code. See #.*INTEG-JCS-CODE* (p. *36*) for details.

Directly threatened asset(s): **D.JCS_CODE**.

T.INTEG-APPLI-DATA The attacker executes an application to alter (part of) another application's data. See #.*INTEG-APPLI-DATA* (p. *36*) for details.

Directly threatened asset(s): **D.APP_I_DATA, D.PIN** and **D.APP_KEYs**.

T.INTEG-JCS-DATA The attacker executes an application to alter (part of) *Java Card System* or API data. See #.*INTEG-JCS-DATA* (p. *36*) for details.

Directly threatened asset(s): **D.API_DATA**, **D.SEC_DATA**, **D.JCS_DATA, D.JCS_KEYs** and **D.CRYPTO**.

Other attacks are in general related to one of the above, and aimed at disclosing or modifying on-card information. Nevertheless, they vary greatly on the employed means and threatened assets, and are thus covered by quite different objectives in the sequel. That is why a more detailed list is given hereafter.

3.4.2.3 Identity Usurpation

T.SID.1 An *applet* impersonates another application, or even the *JCRE*, in order to gain illegal access to some resources of the card or with respect to the end user or the terminal. See #.*SID* (p. *38*) for details.

Directly threatened asset(s): **D.SEC_DATA** (other assets may be jeopardized should this attack succeed, for instance, if the identity of the *JCRE* is usurped), **D.PIN**, **D.APP_KEYs** and **D.JCS_KEYs**

T.SID.2 The attacker modifies the identity of the privileged roles. See #.*SID* (p. *38*) for further details.

Directly threatened asset(s): **D.SEC_DATA** (any other asset may be jeopardized should this attack succeed, depending on whose identity was forged).

3.4.2.4 Unauthorized Execution

T.EXE-CODE.1 An *applet* performs an unauthorized **execution** of a **method**. See *#.EXE-JCS-CODE* (p. 36) and *#.EXE-APPLI-CODE* (p. *36*) for details.

Directly threatened asset(s): **D.APP_CODE**.

T.EXE-CODE.2 An *applet* performs an unauthorized **execution** of a **method fragment** or **arbitrary data**. See *#.EXE-JCS-CODE* (p. *36*) and *#.EXE-APPLI-CODE* (p. *36*) for details.

Directly threatened asset(s): **D.APP_CODE**.

T.NATIVE An *applet* tries to execute a native method to bypass some security function such as the *firewall*. A Java Card technology-based *applet* ("Java Card applet") can only access native methods indirectly (*O.NATIVE)* that is, through an API which is assumed to be secure (*A.NATIVE*). In addition to this, the bytecode verifier also prevents the program counter of an applet to jump into a piece of native code by confining the control flow to the currently executed method (*OE.VERIFICATION*).

3.4.2.5 Denial of Service

T.RESOURCES An attacker prevents correct operation of the *Java Card System* through consumption of some resources of the card: RAM or NVRAM.

Directly threatened asset(s): **D.JCS_DATA**.

3.5 Organizational Security Policies

This section describes the organizational security policies to be enforced with respect to the TOE environment. They are summarized in the following table:

Name	Source	Refined?
OSP.IC_ORG	-	-

Table 3: Organizational Security Policies

3.5.1 Organizational Security Policies

3.5.1.1 OSP on IC Development and Manufacturing (Phases 2 and 3)

OSP.IC_ORG Procedures dealing with physical, personnel, organizational, technical measures for the confidentiality and integrity, of Smart Card Native Operating System (e.g. source code mask and any

associated documents) and IC Manufacturer proprietary information (tools, software, documentation, dies ...) shall exist and be applied in IC development and manufacturing .

Procedures shall also ensure the confidentiality and integrity and information during exchange with the NOS developer.

3.5.2 Organizational Security Policies from [JCSPP]

There are no Organizational Security Policies (OSP) in [JCSPP] for the minimal configuration.

3.6 Security Aspects

This section is partly taken from [JCSPP].

Security aspects are intended to define the main security issues that are to be addressed in the PP and this ST, in a CC-independent way. In addition to this, they also give a semi-formal framework to express the CC security environment and objectives of the TOE. They can be instantiated as assumptions, threats, objectives (for the TOE and the environment), or organizational security policies and are referenced in their definition. For instance, the security aspect #.NATIVE is instantiated in assumption A.NATIVE, threat T.NATIVE and objectives O.NATIVE and OE.NATIVE.

The following sections present several security aspects from [JCSPP] that are relevant for this ST.

3.6.1 Confidentiality

#.CONFID-APPLI-DATA Application data must be protected against unauthorized disclosure. This concerns logical attacks at runtime in order to gain read access to other application's data.

#.CONFID-JCS-CODE *Java Card System* code must be protected against unauthorized disclosure. This concerns logical attacks at runtime in order to gain a read access to executable code, typically by executing an application that tries to read the memory area where a piece of *Java Card System* code is stored.

#.CONFID-JCS-DATA *Java Card System* data must be protected against unauthorized disclosure. This concerns logical attacks at runtime in order to gain a read access to *Java Card System* data. *Java Card System* data includes the data managed by the Java Card runtime environment, the virtual machine and the internal data of Java Card API classes as well.

3.6.2 Integrity

#.INTEG-APPLI-CODE Application code must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain write access to the memory zone where executable code is stored. If the configuration

allows post-issuance application loading, this threat also concerns the modification of application code in transit to the card.

#.INTEG-APPLI-DATA Application data must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain unauthorized write access to application data. If the configuration allows post-issuance application loading, this threat also concerns the modification of application data contained in a *package* in transit to the card. For instance, a *package* contains the values to be used for initializing the static fields of the *package*.

#.INTEG-JCS-CODE *Java Card System* code must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain write access to executable code.

#.INTEG-JCS-DATA *Java Card System* data must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain write access to *Java Card System* data. *Java Card System* data includes the data managed by the Java Card runtime environment, the virtual machine and the internal data of Java Card API classes as well.

3.6.3 Unauthorized Executions

#.EXE-APPLI-CODE Application (byte)code must be protected against unauthorized execution. This concerns **(1)** invoking a method outside the scope of the visibility rules provided by the public/private access modifiers of the Java programming language ([JAVASPEC],§6.6); **(2)** jumping inside a method fragment or interpreting the contents of a data memory area as if it was executable code; **(3)** unauthorized execution of a remote method from the *CAD*.

#.EXE-JCS-CODE *Java Card System* (byte)code must be protected against unauthorized execution. *Java Card System* (byte)code includes any code of the *JCRE* or *API*. This concerns **(1)** invoking a method outside the scope of the visibility rules provided by the public/private access modifiers of the Java programming language ([JAVASPEC],§6.6); **(2)** jumping inside a method fragment or interpreting the contents of a data memory area as if it was executable code. Note that execute access to native code of the *Java Card System* and applications is the concern of #.*NATIVE*.

#.FIREWALL The *Java Card System* shall ensure controlled sharing of class instances[8], and isolation of their data and code between *package*s (that is, controlled execution contexts). **(1)** An *applet* shall neither read, write nor compare a piece of data belonging to an *applet* that is not in the same context, nor execute one of the methods of an applet in another context without its authorization.

#.NATIVE Because the execution of native code is outside of the TOE Scope Control (TSC), it must be secured so as to not provide ways to bypass the TSFs. No untrusted native code may reside on the card. Loading of

[8] This concerns in particular the arrays, which are considered as instances of the Object class in the Java programming language.

native code, which is as well outside the TSC, is submitted to the same requirements. Should native software be privileged in this respect, exceptions to the policies must include a rationale for the new security framework they introduce.

3.6.4 Bytecode Verification

#.VERIFICATION All bytecode must be verified prior to being executed. Bytecode verification includes **(1)** how well-formed *CAP file* is and the verification of the typing constraints on the bytecode, **(2)** binary compatibility with installed *CAP file*s and the assurance that the export files used to check the *CAP file* correspond to those that will be present on the card when loading occurs.

3.6.4.1 CAP File Verification

Bytecode verification includes checking at least the following properties: **(3)** bytecode instructions represent a legal set of instructions used on the Java Card platform; **(4)** adequacy of bytecode operands to bytecode semantics; **(5)** absence of operand stack overflow/underflow; **(6)** control flow confinement to the current method (that is, no control jumps to outside the method); **(7)** absence of illegal data conversion and reference forging; **(8)** enforcement of the private/public access modifiers for *class* and class members; **(9)** validity of any kind of reference used in the bytecodes (that is, any pointer to a bytecode, class, method, object, local variable, etc actually points to the beginning of piece of data of the expected kind); **(10)** enforcement of rules for binary compatibility (full details are given in [JCVM], [JVM], [BCVWP]). **The actual set of checks performed by the verifier is implementation-dependent, but shall at least enforce all the "must clauses" imposed in [JCVM] on the bytecodes and the correctness of the CAP files' format.**

As most of the actual *JCVM*s do not perform all the required checks at runtime, mainly because smart cards lack memory and CPU resources, **CAP file verification prior to execution is mandatory**. On the other hand, there is no requirement on the precise moment when the verification shall actually take place, as far as it can be ensured that the verified file is not modified thereafter. Therefore, the bytecodes can be verified either before the loading of the file on to the card or before the installation of the file in the card or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time.

Note: In the present case, bytecode verification is performed before loading.

Another important aspect to be considered about bytecode verification and application downloading is, first, the assurance that every package required by the loaded *applet* is indeed on the card, in a binary-compatible version (binary compatibility is explained in [JCVM], §4.4), second, that the export files used to check and link the loaded *applet* have the corresponding correct counterpart on the card.

3.6.4.2 Integrity and Authentication

Verification off-card is useless if the application package is modified afterwards. The usage of cryptographic certifications coupled with the verifier in a secure module is a simple means to prevent any attempt of modification between *package* verification and *package* installation. Once a verification authority has verified the package, it signs it and sends it to the card. Prior to the installation of the package, the card verifies the signature of the package, which authenticates the fact that it has been successfully verified. In addition to this, a secured communication channel is used to communicate it to the card, ensuring that no modification has been performed on it.

Alternatively, the card itself may include a verifier and perform the checks prior to the effective installation of the *applet* or provide means for the bytecodes to be verified dynamically.

Note: In the present case, bytecode verification is performed before loading.

3.6.4.3 Linking and Verification

Beyond functional issues, the *installer* ensures at least a property that matters for security: the loading order shall guarantee that each newly loaded *package* references only *package*s that have been already loaded on the card. The linker can ensure this property because the Java Card platform does not support *dynamic* downloading of classes.

3.6.5 Card Management

#.CARD-MANAGEMENT **(1)** The card manager (CM) shall control the access to card management functions such as the installation, update or deletion of *applet*s. **(2)** The card manager shall implement the card issuer 's policy on the card.

#.INSTALL Installation of a *package* or an *applet* is secure. **(1)** The TOE must be able to return to a safe and consistent state should the installation fail or be cancelled (whatever the reasons). **(2)** Installing an application must have no effect on the code and data of already installed *applet*s. The installation procedure should not be used to bypass the TSFs. In short, it is a secure atomic operation, and free of harmful effects on the state of the other *applet*s. **(3)** The procedure of loading and installing a package shall ensure its integrity and authenticity.

Note: In the present case, applet installation takes place in a secure environment prior to card issuing.

#.SID **(1)** Users and subjects of the TOE must be identified. **(2)** The identity of sensitive users and subjects associated with administrative and privileged roles must be particularly protected; this concerns the *JCRE*, the applets registered on the card, and especially the *default applet* and the *currently selected applet* (and all other active applets in Java Card System 2.2.1). A change of identity, especially standing for an administrative role (like an *applet* impersonating the *JCRE*), is a severe violation of the TOE Security Policy (TSP). Selection controls the access to any data exchange between the TOE and the *CAD* and therefore, must be protected as well. The loading of a *package* or any

exchange of data through the *APDU buffer* (which can be accessed by any applet) can lead to disclosure of keys, application code or data, and so on.

#OBJ-DELETION Deallocation of objects must be secure. **(1)** It should not introduce security holes in the form of references pointing to memory zones that are not longer in use, or have been reused for other purposes. Deletion of collection of objects should not be maliciously used to circumvent the TSFs. **(2)** Erasure, if deemed successful, shall ensure that the deleted class instance is no longer accessible.

#DELETION Deletion of applets must be secure. **(1)** Deletion of installed *applet*s (or *package*s) should not introduce security holes in the form of broken references to garbage collected code or data, nor should they alter integrity or confidentiality of remaining *applet*s. The deletion procedure should not be maliciously used to bypass the TSFs. **(2)** Erasure, if deemed successful, shall ensure that any data owned by the deleted *applet* is no longer accessible (shared objects shall either prevent deletion or be made inaccessible). A deleted *applet* cannot be selected or receive APDU commands. Package deletion shall make the code of the package no longer available for execution.**(3)** Power failure or other failures during the process shall be taken into account in the implementation so as to preserve the TSPs. This does not mandate, however, the process to be atomic. For instance, an interrupted deletion may result in the loss of user data, as long as it does not violate the TSPs.

The deletion procedure and its characteristics (whether deletion is either physical or logical, what happens if the deleted application was the *default applet*, the order to be observed on the deletion steps) are implementation-dependent. The only commitment is that deletion shall not jeopardize the TOE (or its assets) in case of failure (such as power shortage).

Deletion of a single *applet* instance and deletion of a whole *package* are functionally different operations and may obey different security rules. For instance, specific *package*s can be declared to be undeletable (for instance, the Java Card API *package*s), or the dependency between installed *package*s may forbid the deletion (like a *package* using super classes or super interfaces declared in another package).

Note: In the present case, deletion of applets is not allowed.

3.6.6 Services

#.ALARM The TOE shall provide appropriate feedback upon detection of a potential security violation. This particularly concerns the type errors detected by the bytecode verifier, the security exceptions thrown by the *JCVM*, or any other security-related event occurring during the execution of a TSF.

#.OPERATE **(1)** The TOE must ensure continued correct operation of its security functions. **(2))** In case of failure during its operation, the TOE must also return to a well-defined valid state before the next service request.

#.RESOURCES	The TOE controls the availability of resources for the applications and enforces quotas and limitations in order to prevent unauthorized denial of service or malfunction of the TSFs. This concerns both execution (dynamic memory allocation) and installation (static memory allocation) of applications and *packages*.
#.CIPHER	The TOE shall provide a means to the applications for ciphering sensitive data, for instance, through a programming interface to low-level, highly secure cryptographic services. In particular, those services must support cryptographic algorithms consistent with cryptographic usage policies and standards.
#.KEY-MNGT	The TOE shall provide a means to securely manage cryptographic keys. This includes: **(1)** Keys shall be generated in accordance with specified cryptographic key generation algorithms and specified cryptographic key sizes, **(2)** Keys must be distributed in accordance with specified cryptographic key distribution methods, **(3)** Keys must be initialized before being used, **(4)** Keys shall be destroyed in accordance with specified cryptographic key destruction methods.
#.PIN-MNGT	The TOE shall provide a means to securely manage PIN objects. This includes: **(1)** Atomic update of PIN value and try counter, **(2)** No rollback on the PIN-checking function,**(3)** Keeping the PIN value (once initialized) secret (for instance, no clear-PIN-reading function), **(4)** Enhanced protection of PIN's security attributes (state, try counter…) in confidentiality and integrity.
#.SCP	The smart card platform must be secure with respect to the TSP. Then: **(1)** After a power loss or sudden card removal prior to completion of some communication protocol, the *SCP* will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state. **(2)** It does not allow the TSFs to be bypassed or altered and does not allow access to other low-level functions than those made available by the *packages* of the API. That includes the protection of its private data and code (against disclosure or modification) from the *Java Card System*. **(3)** It provides secure low-level cryptographic processing to the *Java Card System*. **(4)** It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism. **(5)** It allows the *Java Card System* to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low–level control accesses (segmentation fault detection). **(6)** It safely transmits low–level exceptions to the TOE (arithmetic exceptions, checksum errors), when applicable. We finally require that **(7)** the IC is designed in accordance with a well-defined set of policies and standards (likely specified in another protection profile), and will be tamper resistant to actually prevent an attacker from extracting or altering security data (like cryptographic keys) by using commonly employed techniques (physical probing and sophisticated analysis of the chip). This especially matters to the management (storage and operation) of cryptographic keys.
Note:	In the present case a certified hardware platform is used (see chapter 2).

#.TRANSACTION	The TOE must provide a means to execute a set of operations atomically. This mechanism must not endanger the execution of the user applications. The transaction status at the beginning of an *applet* session must be closed (no pending updates).

4 Security Objectives

This section defines the security objectives to be achieved for the TOE and the environment.

4.1 Security Objectives for the TOE

The Security Objectives for the TOE are summarized in the following table:

Name	Source	Refined?
O.PROTECT_DATA	-	-
O.SIDE_CHANNEL	-	-
O.OS_DECEIVE	-	-
O.FAULT_PROTECT	-	-
O.PHYSICAL	-	-
O.RND	[PP0002]	no
O.SID	[JCSPP]	no
O.OPERATE	[JCSPP]	yes[9]
O.RESOURCES	[JCSPP]	no
O.FIREWALL	[JCSPP]	no
O.NATIVE	[JCSPP]	no
O.REALLOCATION	[JCSPP]	no
O.SHRD_VAR_CONFID	[JCSPP]	no
O.SHRD_VAR_INTEG	[JCSPP]	no
O.ALARM	[JCSPP]	no
O.TRANSACTION	[JCSPP]	no
O.CIPHER	[JCSPP]	no
O.PIN-MNGT	[JCSPP]	no
O.KEY-MNGT	[JCSPP]	no
O.CARD-MANAGEMENT	[JCSPP]	no (*)
O.SCP.RECOVERY	[JCSPP]	no (*)
O.SCP.SUPPORT	[JCSPP]	no (*)
O.SCP.IC	[JCSPP]	no (*)

Table 4: Security Objectives for the TOE

(*) These Security Objectives for the environment of [JCSPP] are Security Objectives for the TOE in the present evaluation. Therefore, the label changed (O.XYZ instead of OE.XYZ) but not the content (no refinement).

[9] Refinement to cover additional aspects of threat T.OS.Operate not contained in [JCSPP].

4.1.1 Security Objectives for the TOE not contained in [JCSPP]

The security objectives of the TOE must cover the following ***aspects***:

- Maintain the integrity of User Data and of the Smart Card Native Operating System (when being executed/processed and when being stored in the TOE's memories) and
- Maintain the confidentiality of User Data and of the Smart Card Native Operating System (when being processed and when being stored in the TOE's memories), as well as
- Provide access control to execution of the TOE code
- Ensure correct operation of the code and maintain the TOE in a secure state
- Protection of the TOE and associated documentation and environment during development and production phases.

The TOE shall use state of the art technology to achieve the following IT security objectives, and for that purpose, when IC physical security features are used, the specification of those IC physical security features shall be respected.

When IC physical security features are not used, the Security Objectives shall be achieved in other ways.

4.1.1.1 Security Objectives for the complete TOE

O.PROTECT_DATA The TOE shall ensure that sensitive information stored in memories is protected against unauthorized disclosure and any corruption or unauthorized modification.

Moreover, the TOE shall ensure that sensitive information stored in memories is protected against unauthorized access.

The TOE has to provide appropriate security mechanisms to avoid fraudulent access to any sensitive data, such as passwords, cryptographic keys or authentication data.

This is obvious for secret information, but also applies to access controlled information sensitive information means here, any primary assets that can be refined by the TOE developer in the Security Target and need to be protected. For the definition of the assets see sections 3.1, 3.1.1, and 3.1.2.

O.SIDE_CHANNEL The TOE must provide protection against disclosure of primary assets including confidential data (User Data or TSF data) stored and/or processed in the Smart Card IC:

- by measurement and analysis of the shape and amplitude
- by measurement and analysis of the time between events found by measuring signals (for example on the power, clock, or I/O lines).

Especially, the NOS must be designed to avoid interpretations of signals extracted, intentionally or not, from the hardware part of the TOE (for instance, Power Supply, Electro Magnetic emissions).

O.OS_DECEIVE The TOE must guarantee that only secure values are used for its management and operations, especially system flags or cryptographic assets.

Moreover, the integrity of the whole TOE including the NOS must be guaranteed to prevent

- Disclosing/bypassing of the NOS mechanisms, or
- Modifying the expected NOS behavior (for instance, unauthorized code patch, or rewriting).

O.FAULT_PROTECT The TOE must ensure its correct operation even outside the normal operating conditions where reliability and secure operation has not been proven or tested. This is to prevent errors. The environmental conditions may include voltage, clock frequency, temperature, or external energy fields that can be applied on all interfaces of the TOE (physical or electrical).

4.1.1.2 Additional Security Objectives for the IC

The hardware participates to objectives of the complete TOE and has to fulfill a specific objective. This objective must be the result of the state of the art of Integrated Circuit security mechanisms. The precise statement of the corresponding requirements for this objective is made in the ST of the hardware platform, which is compliant to the:

- Smart Card IC Platform Protection Profile [PP0002].

O.PHYSICAL The TOE hardware provides the following protection against physical manipulation of the IC, and prevent

- Reverse-engineering (understanding the design and its properties and functions),
- Physical access to the IC active surface (probing) allowing unauthorized memory content disclosure,
- Manipulation of the hardware security parts (e.g. sensors, cryptographic engine or RNG),
- Manipulation of the IC, including the embedded NOS and its application data (e.g. lock and life cycle status, authentication flags, etc.).

4.1.1.3 Security Objective concerning Random Numbers

The following security objective was taken over from [PP0002]:

O.RND Random Numbers

The TOE will ensure the cryptographic quality of random number generation. For instance random numbers shall not be predictable and shall have a sufficient entropy.

The TOE will ensure that no information about the produced random numbers is available to an attacker since they might be used for instance to generate cryptographic keys.

4.1.2 Security Objectives for the TOE from [JCSPP]

4.1.2.1 Identification

O.SID — The TOE shall uniquely identify every subject (*applet*, or *package*) before granting him access to any service.

4.1.2.2 Execution

O.OPERATE — The TOE must ensure continued correct operation of its security functions. Especially, the TOE must prevent the unauthorized use of TOE or use of incorrect or unauthorized instructions or commands or sequence of commands. See #.*OPERATE* (p *39*) for details.

O.RESOURCES — The TOE shall control the availability of resources for the applications. See #.*RESOURCES* (p *40*) for details.

O.FIREWALL — The TOE shall ensure controlled sharing of data containers owned by *applets* of different *package*s, and between *applets* and the TSFs. See #.*FIREWALL* (p *36*) for details.

O.NATIVE — The only means that the *JCVM* shall provide for an application to execute native code is the invocation of a method of the Java Card API, or any additional API. See #.*NATIVE* (p *36*) for details.

O.REALLOCATION — The TOE shall ensure that the re-allocation of a memory block for the runtime areas of the *JCVM* does not disclose any information that was previously stored in that block.

Application note: To be made unavailable means to be physically erased with a default value. Except for local variables that do not correspond to method parameters, the default values to be used are specified in [JCVM221].

O.SHRD_VAR_CONFID — The TOE shall ensure that any data container that is shared by all applications is always cleaned after the execution of an application. Examples of such shared containers are the APDU buffer, the byte array used for the invocation of the process method of the selected applet, or any public global variable exported by the API.

O.SHRD_VAR_INTEG — The TOE shall ensure that only the currently selected application may grant write access to a data memory area that is shared by all applications, like the APDU buffer, the byte array used for the invocation of the process method of the selected applet, or any public global variable exported by the API. Even though the memory area is

shared by all applications, the TOE shall restrict the possibility of getting a reference to such memory area to the application that has been selected for execution. The selected application may decide to temporarily hand over the reference to other applications at its own risk, but the TOE shall prevent those applications from storing the reference as part of their persistent states.

4.1.2.3 Services

O.ALARM The TOE shall provide appropriate feedback information upon detection of a potential security violation. See #.*ALARM* (p. *39*) for details.

O.TRANSACTION The TOE must provide a means to execute a set of operations atomically. See #.*TRANSACTION* (p. *41*) for details.

O.CIPHER The TOE shall provide a means to cipher sensitive data for applications in a secure way. In particular, the TOE must support cryptographic algorithms consistent with cryptographic usage policies and standards. See #.*CIPHER* (p. *40*) for details.

O.PIN-MNGT The TOE shall provide a means to securely manage PIN objects. See #.*PIN-MNGT* (p. *40*) for details.

Application note: PIN objects may play key roles in the security architecture of client applications. The way they are stored and managed in the memory of the smart card must be carefully considered, and this applies to the whole object rather than the sole value of the PIN. For instance, the try counter's value is as sensitive as that of the PIN.

O.KEY-MNGT The TOE shall provide a means to securely manage cryptographic keys. This concerns the correct generation, distribution, access and destruction of cryptographic keys. See #.*KEY-MNGT* (p. *40*).

Application note: *O.*KEY-MNGT, *O.*PIN-MNGT, *O.*TRANSACTION and *O.*CIPHER are actually provided to *applet*s in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to *applet*s; those may be built on top of the Java Card API or independently.

Note: For this Java Card such libraries do not exist. All necessary functionality is implemented by the TOE.

4.1.2.4 Card Management

The TOE Security Objective for the card manager is a Security Objective for the environment in [JCSPP]. In the present case the card manager belongs to the TOE and the corresponding Security Objective is listed here.

O.CARD-MANAGEMENT The card manager shall control the access to card management functions such as the installation, update or deletion of *applet*s. It shall also implement the card issuer's policy on the card.

4.1.2.5 Smart Card Platform

These TOE Security Objectives for the smart card platform are Security Objectives for the environment in [JCSPP]. In the present case the certified smart card platform belongs to the TOE and the corresponding Security Objectives are listed here.

O.SCP.RECOVERY If there is a loss of power, or if the smart card is withdrawn from the CAD while an operation is in progress, the *SCP* must allow the TOE to eventually complete the interrupted operation successfully, or recover to a consistent and secure state (#.*SCP.1*).

O.SCP.SUPPORT The *SCP* shall provide functionalities that support the well-functioning of the TSFs of the TOE (avoiding they are bypassed or altered) and by controlling the access to information proper of the TSFs. In addition, the smart card platform should also provide basic services which are required by the runtime environment to implement security mechanisms such as atomic transactions, management of persistent and transient objects and cryptographic functions. These mechanisms are likely to be used by security functions implementing the security requirements defined for the TOE. See #.*SCP.2-5* (p.40).

O.SCP.IC The *SCP* shall possess IC security features. See #.*SCP.7* (p.*40*).

4.2 Security Objectives for the Environment

The Security Objectives for the Environment are summarized in the following table:

Name	Source	Refined?
OE.DEV_NOS	-	-
OE.DEL_NOS	-	-
OE.IC_ORG	-	-
OE.DLV_PROTECT	-	-
OE.DLV_DATA	-	-
OE.TEST_OPERATE	-	-
OE.USE_DIAG	-	-
OE.USE_KEYS	-	-
OE.NATIVE	[JCSPP]	no
OE.NO-DELETION	[JCSPP]	no
OE.NO-INSTALL	[JCSPP]	no
OE.VERIFICATION	[JCSPP]	no

Table 5: Security Objectives for the environment

4.2.1 Security Objectives for the Environment not contained in [JCSPP]

4.2.1.1 Objectives on Phase 1

OE.DEV_NOS The Smart Card NOS shall be designed in a secure manner, by using exclusively software development tools (compilers assemblers, linkers, simulators, etc.) and software-hardware integration testing tools (emulators) that will result in the integrity of program and data.

The Native Operating System developer shall use established procedures to control storage and usage of the classified development tools and documentation, suitable to maintain the integrity and the confidentiality of the assets of the TOE.

It must be ensured that tools used for the generation of the TOE are only delivered and accessible to the parties authorized personnel. It must be ensured that confidential information on defined assets is only delivered to the parties authorized personnel on a need to know basis.

OE.DEL_NOS The Smart Card Native Operating System and related data must be delivered from the Smart Card Native Operating System developer (phase 1) to the IC designer through a trusted delivery and verification procedure that shall be able to maintain the integrity of the software and its confidentiality, if applicable.

Initialization Data shall be accessible only by authorized personnel (physical, personnel, organizational, technical procedures).

Samples used to run tests shall be accessible only by authorized personnel.

4.2.1.2 Objective on Phases 2 and 3

OE.IC_ORG Procedures dealing with physical, personnel, organizational, technical measures for the confidentiality and integrity, of Smart Card Native Operating System (e.g. source code mask and any associated documents) and IC Manufacturer proprietary information (tools, software, documentation, dice ...) shall exist and be applied in IC development and manufacturing.

Procedures shall exist to ensure the protection of IC sensitive information during exchange with the NOS developer.

4.2.1.3 Objectives on the TOE Delivery Process (Phases 3 to 7)

OE.DLV_PROTECT Procedures shall ensure protection of TOE material/information under delivery including the following objectives:

- o Non-disclosure of any security relevant information, identification of the element under delivery,
- o Meet confidentiality rules (confidentiality level, transmittal form, reception acknowledgement),
- o Physical protection to prevent external damage, secure storage and handling procedures (including rejected TOE's),
- o Traceability of TOE during delivery including the following parameters: origin and shipment details reception, reception acknowledgement, location material/information.

Procedures shall ensure that corrective actions are taken in case of improper operation in the delivery process (including if applicable any non-conformance to the confidentiality convention) and highlight all non-conformance to this process.

Procedures shall ensure that people (shipping department, carrier, reception department) dealing with the procedure for delivery have got the required skill, training and knowledge to meet the procedure requirements and be able to act fully in accordance with the above expectations.

4.2.1.4 Objectives on Delivery to Phases 4, 5 and 6

OE.DLV_DATA The TOE sensitive Data and documentation must be delivered to either the IC packaging manufacturer, to the Card Manufacturer, or to the Personalizer through a trusted delivery and verification procedure that shall be able to maintain the integrity and confidentiality of the TOE sensitive Data.

4.2.1.5 Objectives on Phases 4 to 6

OE.TEST_OPERATE Appropriate functionality testing of the TOE shall be used in phases 4 to 6.

During all manufacturing and test operations, security procedures shall be used through phases 4, 5 and 6 to maintain confidentiality and integrity of the TOE manufacturing and test data.

4.2.1.6 Objectives on Phase 7

OE.USE_DIAG Secure TOE communication protocols shall be supported and used by the environment.

OE.USE_KEYS	During the TOE usage, the terminal or system in interaction with the TOE, shall ensure the protection (integrity and confidentiality) of their own keys by operational means and/or procedures. Application note: Objectives for the TOE environment are usually not satisfied by the TOE Security Functional Requirements. The TOE development and manufacturing environment (phases 1 to 3) is in the scope of this ST. These phases are under the TOE developer scope of control. Therefore, the objectives for the environment related to phase 1 to 3 are covered by Assurance measures, which are materialized by documents, process and procedures evaluated through the TOE evaluation process. The `product usage phases` (phase 4 to 7) are not in the scope of the evaluation. During these phases, the TOE is no more under the developer control. In this environment, the TOE protects itself with its own Security functions. But some additional usage recommendation must also be followed in order to ensure that the TOE is correctly and securely handled, and that shall be not damaged or compromised. This ST assumes (A.DLV_DATA, A.TEST_OPERATE, A.USE_DIAG, A.USE_KEYS) that users handle securely the TOE and related Objectives for the environment are defined (OE.DLV_DATA, OE.TEST_OPERATE, OE.USE_DIAG, OE.USE_KEYS)

4.2.2 Security Objectives for the Environment from [JCSPP]

OE.NATIVE	Those parts of the APIs written in native code as well as any pre-issuance native application on the card shall be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p.36) for details.
Note:	The Security Objectives from [JCSPP] for the environment *OE.SCP.RECOVERY*, *OE.SCP.SUPPORT*, and *O.SCP.IC* are listed as TOE security objectives for the TOE in section 4.1.2.5 as the Smart Card Platform belong to the TOE for this evaluation.
Note:	The Security Objective from [JCSPP] for the environment *OE.CARD-MANAGEMENT* is listed as TOE security objective for the TOE in section 4.1.2.4 as the Card Manager belongs to the TOE for this evaluation.
OE.NO-DELETION	No installed *applet*s (or *package*s) shall be deleted from the card.

OE.NO-INSTALL There is no post-issuance installation of *applet*s. Installation of *applet*s is secure and shall occur only in a controlled environment in the pre-issuance phase.

The objectives *OE.NO-INSTALL* and *OE.NO-DELETION* have been included so as to describe procedures that shall contribute to ensure that the TOE will be used in a secure manner. Moreover, they have been defined in accordance with the environmental assumptions they uphold (actually, they are just a reformulation of the corresponding assumptions). The ***NO-DELETION*** and ***NO-INSTALL*** (assumptions and objectives) constitute the explicit statement that the Minimal configuration corresponds to that of a closed card (no code can be loaded or deleted once the card has been issued). It is not evident that these objectives should be carried out by using IT means.

OE.VERIFICATION All the bytecodes shall be verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time. See #.*VERIFICATION* (p.*37*) for details.

5 IT Security Requirements

5.1 TOE Security Functional Requirements

This section defines the functional requirements for the TOE using only functional requirements components drawn from the CC part 2 except four additional SFR (FCS_RND.1, FMT_LIM.1, FMT_LIM.2, FPT_EMSEC.1) not contained in CC part 2. The functional requirements were taken from [JCSPP] (sections 5.1.1-5.1.6) and newly defined (section 5.1.7).

The permitted operations (assignment, iteration, selection and refinement) of the SFR related to [CC] are printed in ***bold and italics*** type. Completed operations related to the PP are additionally marked within **[ ]**. Editorial changes are printed in *italics*. If appropriate, an explanatory note on the operation is given, e. g. to justify a difference to the SFR as defined in the underlying PP.

The prefix used to introduce objects is “OB”. This was done to avoid confusions with the TOE objectives even though [JCSPP] uses the same prefix “OB” for TOE objectives and objects.

The assurance level for this ST is **EAL4** augmented by **ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3**. The minimum strength level for the TOE security functions is **SOF-high**.

The following table gives an overview of the used SFR for the TOE from part 2 [CC]: (see also table 14 in section 8.2.1.3 which shows all SFR, iterations and dependencies)

No.	Class / Component	Name	Hierar.	Dependency
	FAU	**Security audit**		
1.	FAU_ARP.1	Security alarms	no	FAU_SAA.1
2.	FAU_SAA.1	Potential violation analysis	no	FAU_GEN.1
	FCS	**Cryptographic support**		
3.	FCS_CKM.1	Cryptographic key generation	no	[FCS_CKM.2 or FCS_COP.1] FCS_CKM.4, FMT_MSA.2
4.	FCS_CKM.2	Cryptographic key distribution	no	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2
5.	FCS_CKM.3	Cryptographic key access	no	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2
6.	FCS_CKM.4	Cryptographic key destruction	no	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FMT_MSA.2

No.	Class / Component	Name	Hierar.	Dependency
7.	FCS_COP.1	Cryptographic operation	no	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2
	FDP	**User data protection**		
8.	FDP_ACC.1	Subset access control	no	FDP_ACF.1
9.	FDP_ACC.2	Complete access control	FDP_ACC.1	FDP_ACF.1
10.	FDP_ACF.1	Security attribute based access control	no	FDP_ACC.1, FMT_MSA.3
11.	FDP_ETC.1	Export of user data without security attributes	no	[FDP_ACC.1 or FDP_IFC.1]
12.	FDP_IFC.1	Subset Information flow control	no	FDP_IFF.1
13.	FDP_IFF.1	Simple security attributes	no	FDP_IFC.1, FMT_MSA.3
14.	FDP_ITC.1	Import of user data without security attributes	no	[FDP_ACC.1 or FDP_IFC.1] FMT_MSA.3
15.	FDP_RIP.1	Subset residual information protection	no	no
16.	FDP_ROL.1	Basic rollback	no	[FDP_ACC.1 or FDP_IFC.1]
17.	FDP_SDI.2	Stored data integrity monitoring and action	FDP_SDI.1	no
	FIA	**Identification and authentication**		
18.	FIA_AFL.1	Authentication failure handling	no	FIA_UAU.1
19.	FIA_ATD.1	User attribute definition	no	no
20.	FIA_UAU.1	Timing of authentication	no	FIA_UID.1
21.	FIA_UAU.3	Unforgeable authentication	no	no
22.	FIA_UAU.4	Single-use authentication mechanisms	no	no
23.	FIA_UID.1	Timing of identification	no	no
24.	FIA_UID.2	User identification before any action	FIA_UID.1	no
25.	FIA_USB.1	User-subject binding	no	FIA_ATD.1
	FMT	**Security management**		
26.	FMT_MSA.1	Management of security attributes	no	[FDP_ACC.1 or FDP_IFC.1] FMT_SMF.1, FMT_SMR.1
27.	FMT_MSA.2	Secure security attributes	no	ADV_SPM.1 [FDP_ACC.1 or FDP_IFC.1] FMT_MSA.1, FMT_SMR.1
28.	FMT_MSA.3	Static attribute initialization	no	FMT_MSA.1, FMT_SMR.1

No.	Class / Component	Name	Hierar.	Dependency
29.	FMT_MTD.1	Management of TSF data	no	FMT_SMF.1, FMT_SMR.1
30.	FMT_MTD.3	Secure TSF data	no	ADV_SPM.1, FMT_MTD.1
31.	FMT_SMF.1	Specification of Management Functions	no	no
32.	FMT_SMR.1	Security roles	no	FIA_UID.1
	FPR	**Privacy**		
33.	FPR_UNO.1	Unobservability	no	no
	FPT	**Protection of the TSF**		
34.	FPT_AMT.1	Abstract machine testing	no	no
35.	FPT_FLS.1	Failure with preservation of secure state	no	ADV_SPM.1
36.	FPT_PHP.1	Passive detection of physical attack	no	no
37.	FPT_PHP.3	Resistance to physical attack	no	no
38.	FPT_RVM.1	Reference mediation	no	no
39.	FPT_SEP.1	TSF domain separation	no	no
40.	FPT_RCV.3	Trusted Recovery	FPT_RCV.2	AGD_ADM.1, ADV_SPM.1
41.	FPT_RCV.4	Trusted Recovery	no	ADV_SPM.1
42.	FTP_TDC.1	Inter-TSF basic TSF data consistency	no	no
43.	FPT_TST.1	TSF testing	no	FPT_AMT.1
	FRU	**Resource utilization**		
44.	FRU_FLT.2	Limited fault tolerance	FRU_FLT.1	FPT_FLS.1
	FTP	**Trusted path/channels**		
45.	FTP_ITC.1	Inter-TSF trusted channel	no	no

Table 6: TOE security functionality requirements

Additionally, the extended SFR FCS_RND.1 has been taken over from [PP0002] and FMT_LIM.1, FMT_LIM.2 and FPT_EMSEC.1 have been taken over from the certified (BSI-PP-0017) Protection Profile *Machine Readable travel Document with "ICAO Application", Basic Access Control* [PP0017].

5.1.1 Firewall Policy

5.1.1.1 FDP_ACC.2: Complete Access Control

FDP_ACC.2.1/FIREWALL The TSF shall enforce the ***FIREWALL access control SFP*** *on* ***S.PACKAGE, S.JCRE, OB.JAVAOBJECT*** and all operations among subjects and objects covered by the SFP.

Subjects (prefixed with an "*S*") and objects (prefixed with an "*OB*") covered by this policy are:

Subject / Object	Description
S.PACKAGE	Any *package*, which is the security unit of the firewall policy.
S.JCRE	The *JCRE*. This is the process that manages *applet* selection and de-selection, along with the delivery of *APDU*s from and to the smart card device. *This subject is unique.*
OB.JAVAOBJECT	Any object. Note that KEYS, PIN, arrays and *applet* instances are specific objects in the Java programming language.

Operations (prefixed with "*OP*") of this policy are described in the following table. Each operation has a specific number of parameters given between brackets, among which there is the "**accessed object**", the first one, when applicable. Parameters may be seen as security attributes that are under the control of the subject performing the operation.

Operation	Description
OP.ARRAY_ACCESS(OB.JAVAOBJECT, field*)*	Read/Write an array component.
OP.INSTANCE_FIELD(OB.JAVAOBJECT, field*)*	Read/Write a field of an instance of a class in the Java programming language
OP.INVK_VIRTUAL(OB.JAVAOBJECT, method, arg_1,…*)*	Invoke a virtual method (either on a class instance or an array object)
OP.INVK_INTERFACE(OB.JAVAOBJECT, method, arg_1,…*)*	Invoke an *interface* method.
OP.THROW(OB.JAVAOBJECT)	Throwing of an object (**athrow**).
OP.TYPE_ACCESS(OB.JAVAOBJECT, class*)*	Invoke **checkcast** or **instanceof** on an object.
OP.JAVA(…)	Any access in the sense of [JCRE221], §6.2.8. In our formalization, this is one of the preceding operations.
*OP.CREATE(*Sharing, LifeTime*)*	Creation of an object (**new** or **makeTransient** call).

Note that accessing array's components of a **static** array, and more generally fields and methods of **static** *objects,* is an access to the corresponding *OB.JAVAOBJECT*.

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

5.1.1.2 FDP_ACF.1 Security Attribute based Access Control

See FMT_MSA.1 for more information about security attributes.

FDP_ACF.1.1/FIREWALL The TSF shall enforce the ***FIREWALL access control SFP*** to objects based on *the following*: ***the security attributes of the covered subjects and objects contained in the following tables.***

Editorial changes: 1.) The word "*the following*" results from final interpretation FI103 which was not considered in [JCSPP].

2.) The assignment contained in [JCSPP] "***(1) the security attributes of the covered subjects and objects, (2) the currently active context and (3) the SELECTed applet context***" has been changed according to F103, because attributes must be assigned to either subjects or objects.

The following table describes which security attributes are attached to which subject/object of our policy.

Subject/Object	Attributes
S.PACKAGE	Context
S.JCRE	None
OB.JAVAOBJECT	Sharing, Context, LifeTime, SELECTed applet Context[10]

The following table describes the possible values for each security attribute.

Name	Description
Context	*Package AID*, or "JCRE"
Sharing	Standard, SIO, JCRE entry point, or global array
LifeTime	**CLEAR_ON_DESELECT** or **PERSISTENT**.[11]
SELECTed applet Context	*Package AID*, or "None"

In the case of an array type, we state that fields are components of the array ([JVM], §2.14, §2.7.7), as well as the length; the only methods of an array object are those inherited from the **Object** *class*.

The Sharing attribute defines four categories of objects:
– Standard ones, whose both fields and methods are under the firewall policy,

[10] The security attribute SELECTed applet Context was assigned to object OB.JAVAOBJECT according to final interpretation FI103.

[11] *Transient objects* of type **CLEAR_ON_RESET** behave like persistent objects in that they can be accessed only when the currently active context is the object's context.

- Shareable interface Objects (SIO), which provide a secure mechanism for inter-applet communication,
- *JCRE entry points* (Temporary or Permanent), who have freely accessible methods but protected fields,
- Global arrays, having both unprotected fields (including components; refer to JavaCardClass discussion above) and methods.

When a new object is created, it is associated with the currently active context. But the object is owned by the *applet* instance within the currently active context when the object is instantiated ([JCRE221], §6.1.2). An object is owned by an *applet* instance, by the *JCRE* or by the *package* library where it has been defined (these latter objects can only be arrays that initialize static fields of packages).

Finally both "the currently active context" and "the SELECTed applet context" are security attributes internal to the VM, that is, not attached to any specific object or subject of the Security Policy Model ("SPM"). They are TSF data that play a role in the SPM.

> ([JCRE221], Glossary) *Currently selected applet***.** The JCRE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command with this applet's *AID*, the JCRE makes this applet the currently selected applet. The JCRE sends all APDU commands to the currently selected applet.

While the expression "selected applet" refers to a specific installed applet, the relevant aspect to the policy is the *context* of the selected *applet*; that is why the associated security attribute is a *package AID*.

> ([JCRE221] §6.1.1) At any point in time, there is only **one active context** within the VM (this is called the *currently active context*).

This should be identified in our model with the acting *S.PACKAGE*'s context (see "*Currently context*" in the glossary). This value is in one-to-one correspondence with *AIDs* of *package*s (except for the JCRE context, of course), which appears in the model in the "Context" attribute of both subjects and objects of the policy. The reader should note that the invocation of **static** methods (or access to a **static** field) is not considered by this policy, as there are no firewall rules. They have no effect on the active context as well and the "acting package" is not the one to which the **static** method belongs in this case.

FDP_ACF.1.2/FIREWALL The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed by the ***FIREWALL SFP****:*

R.JAVA.1 ([JCRE221]§6.2.8) An *S.PACKAGE* may freely perform any of *OP.ARRAY_ACCESS,OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW* or *OP.TYPE_ACCESS* upon any *OB.JAVAOBJECT* whose Sharing attribute has value **"*JCRE entry point*"** or **"global array".**

R.JAVA.2 ([JCRE221]§6.2.8) An *S.PACKAGE* may freely perform any of *OP.ARRAY_ACCESS,OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE* or *OP.THROW* upon any *OB.JAVAOBJECT* whose Sharing attribute has value **"Standard"** and whose Lifetime attribute has value **"PERSISTENT"** only if *OB.JAVAOBJECT*'s Context attribute has the same value as the active context.

R.JAVA.3 ([JCRE221]§6.2.8.10) An *S.PACKAGE* may perform *OP.TYPE_ACCESS* upon an *OB.JAVAOBJECT* whose Sharing attribute has value **"SIO"** only if *OB.JAVAOBJECT* is being cast into (**checkcast**) or is being verified as being an instance of (**instanceof**) an *interface* that extends the **Shareable** *interface*.

R.JAVA.4 ([JCRE221]§6.2.8.6) An *S.PACKAGE* may perform *OP.INVK_INTERFACE* upon an *OB.JAVAOBJECT* whose Sharing attribute has the value **"SIO"** only if the invoked *interface* method extends the **Shareable** *interface*.

R.JAVA.5 An *S.PACKAGE* may perform an *OP.CREATE* only if the value of the Sharing parameter[12] is "Standard".

At last, rules governing access to and creation of *OB.JAVAOBJECT*s by *S.JCRE* are essentially implementation-dependent (however, see ***FDP_ACF.1.3/FIREWALL***.)

FDP_ACF.1.3/FIREWALL The TSF shall explicitly authorize access of subjects to objects based on the following additional *rule*[13]:

The subject* S.JCRE *can freely perform* OP.JAVA(…) *and* OP.CREATE*, with the exception given in* FDP_ACF.1.4/FIREWALL*.

FDP_ACF.1.4/FIREWALL The TSF shall explicitly deny access of subjects to objects based on the ***rules***:

1) Any subject with** OP.JAVA **upon an** OB.JAVAOBJECT **whose LifeTime attribute has value "CLEAR_ON_DESELECT" if OB.JAVAOBJECT's Context attribute is not the same as the SELECTed applet Context.

2) Any subject with** OP.CREATE* and ***a "CLEAR_ON_DESELECT" LifeTime parameter if the active context is not the same as the SELECTed applet Context.

Application note: The deletion of *applet*s may render some *OB.JAVAOBJECT* inaccessible, and the JCRE may be in charge of this aspect. This can be done, for instance, by ensuring that references to objects belonging to a deleted application are considered as a **null** reference. Such a mechanism is implementation-dependent.

Note: For this Java Card no applet deletion is possible, therefore this application note is not relevant for this evaluation.

[12] For this operation, there is no accessed object; the "Sharing value" thus refers to the parameter of the operation. This rule simply enforces that shareable transient objects are not allowed. Note: parameters can be seen as security attributes whose value is under the control of the subject. For instance, during the creation of an object, the JavaCardClass attribute's value is chosen by the creator.

[13] Editorial Change: The word "rules" in CC part 2 was replaced by "rule" as proposed in [JCSPP], because only one rule is added.

5.1.1.3 FDP_IFC.1 Subset Information Flow Control

FDP_IFC.1.1/JCVM The TSF shall enforce the ***JCVM information flow control SFP*** on ***the following subjects, information and operations***.

Subjects[14] (prefixed with an "*S*") and information (prefixed with an "*I*") covered by this policy are:

Subject/Information	Description
S.LOCAL	Operand stack of a JCVM frame, or local variable of a JCVM frame containing an object or an array of references.
S.MEMBER	Any object's field, **static** field or array position.
I.DATA	JCVM Reference Data: *objectref addresses of temporary JCRE Entry Point objects and global arrays*.

There is a unique operation in this policy:

Operation	Description
OP.PUT(S_1, S_2, I)	Transfer a piece of information *I* from S_1 to S_2.

Application note: References of temporary *JCRE entry points*, which cannot be stored in *class* variables, instance variables or array components, are transferred from the internal memory of the *JCRE* (TSF data) to some stack through specific APIs (*JCRE* owned exceptions) or *JCRE* invoked methods (such as the **process(APDU apdu)**); these are causes of *OP.PUT(S_1,S_2,I)* operations as well.

5.1.1.4 FDP_IFF.1 Simple Security Attributes

FDP_IFF.1.1/JCVM The TSF shall enforce the ***JCVM information flow control SFP*** based on the following types of subject and information security attributes: ***subjects: S.LOCAL, S.MEMBER; information I.DATA; attribute: the currently active context.***

Refinement: Due to application of final interpretation FI104 the assignment must contain the list of subjects, information, and corresponding security attributes.

FDP_IFF.1.2/JCVM The TSF shall permit an information flow between a controlled subject and controlled information *through*[15] a controlled operation if the following *rule*[16] holds:

[14] Information flow policies control the flow of information between "subjects". This is a purely terminological choice; those "subjects" can merely be passive containers. They are not to be confused with the "active entities" of access control policies.

[15] Editorial change: The word "via" was replaced by "through" as proposed in [JCSPP].

[16] Editorial change: The word "rules" in CC part 2 was replaced by "rule", as only one rule is added.

An operation *OP.PUT(S_1, S.MEMBER, I)* is allowed if and only if the active context is "JCRE"; other *OP.PUT* operations are allowed regardless of the active context's value.

FDP_IFF.1.3/JCVM The TSF shall enforce **[*no additional information flow control SFP rules*]**.

FDP_IFF.1.4/JCVM The TSF shall provide the following **[*no additional SFP capabilities*]**.

FDP_IFF.1.5/JCVM The TSF shall explicitly authorize an information flow based on the following rules: **[*none*]**.

FDP_IFF.1.6/JCVM The TSF shall explicitly deny an information flow based on the following rules: **[*none*]**

Application note: the storage of temporary *JCRE*-owned objects' references is runtime-enforced ([JCRE221], §6.2.8.1-3).

Note that this policy essentially applies to the execution of bytecode. Native methods[17], the JCRE itself and possibly some API methods can be granted specific rights or limitations through the ***FDP_IFF.1.3/JCVM*** to ***FDP_IFF.1.6/JCVM*** elements. The way the virtual machine manages the transfer of values on the stack and local variables (returned values, uncaught exceptions) from and to internal registers is implementation-dependent. For instance, a returned reference, depending on the implementation of the stack frame, may transit trough an internal register prior to being pushed on the stack of the invoker. The **areturn** bytecode would cause more than one *OP.PUT* operation under this scheme.

5.1.1.5 FDP_RIP.1 Subset Residual Information Protection

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

Application note: The semantics of the Java programming language requires for any object field and array position to be initialized with default values when the resource is allocated [JVM],§2.5.1.

5.1.1.6 FMT_MSA.1 Management of Security Attributes

(See ***FMT_SMR.1.1/JCRE*** for the roles)

FMT_MSA.1.1/JCRE The TSF shall enforce the ***FIREWALL access control SFP and the JCVM information flow control SFP*** to restrict the ability to ***modify*** *the* **active context and the SELECTed applet Context** *security attributes*[18] to **the JCRE (***S.JCRE***).**

Application note: The modification of the active context as well as that of the selected applet should be performed in accordance with the rules given in [JCRE221], §4 and [JCVM221], §3.4.

[17] Note: For this TOE, they are no native methods.

[18] Editorial change for better readability of the sentence.

5.1.1.7 FMT_MSA.2 Secure Security Attributes

FMT_MSA.2.1/JCRE The TSF shall ensure that only secure values are accepted for security attributes.

Application note: For instance, secure values conform to the following rules:

- The Context attribute of a **.JAVAOBJECT*[19] must correspond to that of an installed *applet* or be "JCRE".
- An *OB.JAVAOBJECT* whose Sharing attribute is a *JCRE entry point* or a global array necessarily has "JCRE" as the value for its Context security attribute.
- An *OB.JAVAOBJECT* whose Sharing attribute value is a global array necessarily has "array of primitive Java Card System type" as a JavaCardClass security attribute's value.
- Any *OB.JAVAOBJECT* whose Sharing attribute value is not "Standard" has a **PERSISTENT**-LifeTime attribute's value.
- Any *OB.JAVAOBJECT* whose LifeTime attribute value is not **PERSISTENT** has an array type as JavaCardClass attribute's value.

Application note: The above rules are given as examples only. For instance, the last two rules are motivated by the fact that the Java Card API defines only transient arrays factory methods. Future versions may allow the creation of transient objects belonging to arbitrary classes; such evolution will naturally change the range of "secure values" for this component.

5.1.1.8 FMT_MSA.3 Static Attribute Initialization

FMT_MSA.3.1/FIREWALL The TSF shall enforce the ***FIREWALL access control SFP*** *and the* ***JCVM information flow control SFP*** to provide ***restrictive*** default values for security attributes that are used to enforce the SFP.

Application note: Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (*FMT_MSA.1/JCRE*). At the creation of an object (*OP.CREATE*), the newly created object, assuming that the operation is permitted by the SFP, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([JCRE221], §6.1.2). There is one default value for the *SELECTed applet Context* that is the *default applet identifier's Context*, and one default value for the *active context*, that is "*JCRE*".

Application note: There is no security attribute attached to subjects or information for this information flow policy. However, this is the JCRE who controls the currently active context. Moreover, the knowledge of which reference corresponds to a temporary entry point object or a global array and which does not is solely available to the JCRE (and the virtual machine).

FMT_MSA.3.2/FIREWALL The TSF shall allow the ***following role(s)*** to specify alternative initial values to override the default values when an object or information is created*: none*[20].

[19] Either subject or object.

[20] Editorial Change: The sentence was changed for better readability.

Application note: The intent is that none of the identified roles has privileges with regard to the default values of the security attributes. Notice that creation of objects is an operation controlled by the ***FIREWALL SFP***; the latitude on the parameters of this operation is described there. The operation shall fail anyway if the created object would have had security attributes whose value violates ***FMT_MSA.2.1/JCRE***.

5.1.1.9 FMT_SMR.1 Security roles

FMT_SMR.1.1/JCRE The TSF shall maintain the roles: ***the JCRE***.

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

5.1.1.10 FPT_SEP.1 TSF domain separation

FPT_SEP.1.1 The TSF shall maintain a security domain for its own execution that protects it from interference and tampering by untrusted subjects.

FPT_SEP.1.2 The TSF shall enforce separation between the security domains of subjects in the TSC.

Application note: By security domain it is intended “execution context” which should not be confused with other meanings of “security domains”.

5.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

5.1.2.1 FCS_CKM.1 Cryptographic KEY Generation

The whole set of cryptographic algorithms is generally not implemented because of limited memory resources and/or limitations due to exportation. Therefore, the following requirement should only apply to the implemented subset.

FCS_CKM.1.1 The TSF shall generate cryptographic KEYS in accordance with a specified cryptographic KEY generation algorithm **[*none*]** and specified cryptographic KEY sizes **[*DES: 112, 168 Bit, RSA: 1024 - 2368 Bit, AES: 128, 192, 256 Bit*]** that meet the following: **[*none*]**.

Application note: The keys can be generated and diversified in accordance with [JCAPI221] specification in *class*es **KeyBuilder** and **KeyPair** (at least Session key generation).

5.1.2.2 FCS_CKM.2 Cryptographic KEY Distribution

FCS_CKM.2.1 The TSF shall distribute cryptographic KEYS in accordance with a specified cryptographic KEY distribution method **[*methods: setKey for DES and AES, as well as setExponent and setModulus for RSA*]** that meets the following: **[*[JCAPI221]*]**.

5.1.2.3 FCS_CKM.3 Cryptographic KEY Access

FCS_CKM.3.1 The TSF shall perform **[*management of DES, AES, and RSA-keys*]** in accordance with a specified cryptographic key access method

[*methods/commands defined in packages javacard.security and javacardx.crypto of [JCAPI221]*] that meets the following: **[*[JCAPI221]*]**.

Application note: The keys can be accessed in accordance with [JCAPI221] in *class* **Key**.

5.1.2.4 FCS_CKM.4 Cryptographic KEY Destruction

FCS_CKM.4.1 The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method **[*physically overwriting the keys by method clearKey of [JCAPI221]*]** that meets the following: **[none]**.

Application note: The keys are reset in accordance with [JCAPI221] in *class* **Key** with the method **clearKey()**. Any access to a cleared key attempting to use it for ciphering or signing shall throw an exception.

5.1.2.5 FCS_COP.1 Cryptographic Operation

FCS_COP.1/TripleDES

FCS_COP.1.1 The TSF shall perform **[*data encryption and decryption*]** in accordance with a specified cryptographic algorithm **[*Triple-DES in ECB/CBC Mode*]** and cryptographic key size **[*112, 168 Bit*]** that meet the following: **[*FIPS 46-3*]**.

FCS_COP.1/AES

FCS_COP.1.1 The TSF shall perform **[*data encryption and decryption*]** in accordance with a specified cryptographic algorithm **[*AES in ECB/CBC Mode*]** and cryptographic key size **[*128, 192, and 256 Bit*]** that meet the following: **[*FIPS 197*]**.

FCS_COP.1/RSACipher

FCS_COP.1.1 The TSF shall perform **[*data encryption and decryption*]** in accordance with a specified cryptographic algorithm **[*RSA*]** and cryptographic key size **[*1024 - 2368 Bit*]** that meet the following: **[*PKCS#1 v1.5.*]**.

FCS_COP.1/DESMAC

FCS_COP.1.1 The TSF shall perform **[*8 byte MAC generation and verification*]** in accordance with a specified cryptographic algorithm **[*Triple-DES in outer CBC Mode*]** and cryptographic key size **[*112, 168 Bit*]** that meet the following: **[*ISO 9797-1*]**.

FCS_COP.1/RSASignatureISO9796

FCS_COP.1.1 The TSF shall perform **[*digital signature generation and verification*]** in accordance with a specified cryptographic algorithm **[*RSA with SHA-1*]** and cryptographic key size **[*1024 - 2368 Bit*]** that meet the following: **[*ISO 9796-2*]**.

FCS_COP.1/RSASignaturePKCS#1

FCS_COP.1.1 The TSF shall perform **[*digital signature generation and verification*]** in accordance with a specified cryptographic algorithm **[*RSA with SHA-1*]** and cryptographic key size **[*1024 - 2368 Bit*]** that meet the following: **[*PKCS#1 v1.5*]**.

FCS_COP.1/SHA-1

FCS_COP.1.1 The TSF shall perform **[*secure hash computation*]** in accordance with a specified cryptographic algorithm **[*SHA-1*]** and cryptographic key size **[*none*]** that meet the following: **[*FIPS 180-1*]**.

5.1.2.6 FDP_RIP.1 Subset Residual Information Protection

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

Application note: The allocation of a resource to the APDU buffer is typically performed as the result of a call to the **process()** method of an applet.

FDP_RIP.1.1/bArray The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation[21] of the resource from*** the following object*:* ***the bArray object***.

Application note: A resource is allocated to the bArray object when a call to *an* applet's **install()** method is performed*.* There is no conflict with FDP_ROL.1 here because of the bounds on the rollback mechanism (***FDP_ROL.1.2/FIREWALL***): the scope of the rollback does not extend outside the execution of the **install()** method, and the de-allocation occurs precisely right after the return of it.

FDP_RIP.1.1/TRANSIENT The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation***[21] ***of the resource from*** the following objects*: **any transient object**.*

Application note: The events that provoke the de-allocation of a transient object are described in [JCRE221]*, §5.1.*

FDP_RIP.1.1/ABORT The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation***[21] ***of the resource from*** the following objects*: **any reference to an object instance created during an aborted transaction**.*

Application note: The events that provoke the de-allocation of the previously mentioned references are described in [JCRE221], §7.6.3.

FDP_RIP.1.1/KEYS The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation***[21] ***of the resource from*** the following objects: ***the cryptographic buffer (*D.CRYPTO*)***.

Application note: The **javacard.security** & **javacardx.crypto** *package*s do provide secure interfaces to the ***cryptographic buffer*** in a transparent way. See **javacard.security.KeyBuilder** and **Key** *interface* of [JCAPI221].

[21] Editorial Change: The word "deallocation" was replaced by "de-allocation" as proposed in [JCSPP].

Application note: Java Card System 2.1.1 defines no explicit (or implicit) de-allocation of objects, but those caused by the failure of installation or the abortion of a transaction. The only related function for keys is the **clearKey()** method, which does not mandate erasure of the contents of the key (see ***FCS_CKM.4***) nor the behavior of the transaction with respect to this "clearing". ST authors may consider additional security requirements on this topic.

5.1.2.7 FDP_ROL.1 Basic Rollback

FDP_ROL.1.1/FIREWALL The TSF shall enforce ***the FIREWALL access control SFP and the JCVM information flow control SFP*** to permit the rollback of ***OP.JAVA, OP.CREATE*** *on*[22] ***OB.JAVAOBJECTs.***

FDP_ROL.1.2/FIREWALL The TSF shall permit operations to be rolled back within the ***scope of a*** **select()**, **deselect()**, **process()** or **install()** ***call, notwithstanding the restrictions given in*** [JCRE221], §7.7, within the bounds of ***the Commit Capacity*** ([JCRE221], §7.8), ***and those described in*** [JCAPI221].

Application note: Transactions are a service offered by the APIs to *applet*s. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [JCAPI221] (see for instance, PIN-blocking, PIN-checking, update of *Transient object*s).

Application note: The loading and linking of *applet package*s (the installation or registration is covered by ***FDP_ROL.1.1/FIREWALL***) is subject to some kind of rollback mechanism, described in [JCRE221], §10.1.4, but is implementation-dependent.

5.1.3 Card Security Management

The following SFRs are related to the security requirements at the level of the whole card, in contrast to the previous ones, that are somewhat restricted to the TOE alone. For instance, a potential security violation detected by the virtual machine may require a reaction that does not only concern the virtual machine, such as blocking the card (or request the appropriate security module with the power to block the card to perform the operation).

5.1.3.1 FAU_ARP.1 Security Alarms

FAU_ARP.1.1/JCS The TSF shall *throw*[23] ***an exception, lock the card session or reinitialize the Java Card System and its data*** **[no other actions]** upon detection of a potential security violation.

REFINEMENT Potential security violation is refined to one of the following events:

[22] Editorial change: Rewording for better understanding as proposed in [JCSPP].

[23] Editorial change: Rewording for better understanding as proposed in [JCSPP].

- *applet* life cycle[24] inconsistency
- Card tearing (unexpected removal of the Card out of the CAD) and power failure
- Abortion of a transaction in an unexpected context (see (**abortTransaction()**,[JCAPI221] and ([JCRE221], §7.6.2)
- Violation of the Firewall or JCVM SFPs
- Unavailability of resources
- Array overflow
- Other runtime errors related to *applet*'s failure (like uncaught exceptions)
- **Card Manager life cycle state (OP_READY, INITIALIZED, SECURED, CARD_LOCKED, TERMINATED) inconsistency audited through the life cycle checks in all administrative operations and the self test mechanism on start-up.**
- **Abnormal environmental conditions (frequency, voltage, temperature)**
- **Physical tampering**
- **EEPROM failure audited through exceptions in the read/write operations and consistency/integrity check**
- **Corruption of check-summed objects**
- **Illegal access to the previously defined D.JAVA_OBJECT objects audited through the firewall mechanism** [25]

Application note: The thrown exceptions and their related events are described in [JCRE221], [JCAPI221], and [JCVM221].

Application note: The bytecode verification defines a large set of rules used to detect a "potential security violation". The actual monitoring of these "events" within the TOE only makes sense when the bytecode verification is performed on-card.

Application note: Depending on the context of use and the required security level, there are cases where the card manager and the TOE must work in cooperation to detect and appropriately react in case of potential security violation. This behavior must be described in this component. It shall detail the nature of the feedback information provided to the card manager (like the identity of the offending application) and the conditions under which the feedback will occur (any occurrence of the **java.lang.SecurityException** exception).

Application note: The "locking of the card session" may not appear in the policy of the card manager. Such measure should only be taken in case of severe violation detection; the same holds for the re-initialization of the Java Card System. Moreover, the locking should occur when "clean" re-initialization seems to be impossible.

The locking may be implemented at the level of the *Java Card System* as a denial of service (through some systematic "fatal error" message or return value) that lasts up to

[24] Applet life cycle states are INSTALLED, SELECTABLE, LOCKED. In addition to these Application Life Cycle States, the Application may define its own Application dependent states.

[25] The events in printed in bold-style are additional events regarding to [PP0002]

the next “RESET” event, without affecting other components of the card (such as the card manager).

Finally, because the installation of *applet*s is a sensitive process, security alerts in this case should also be carefully considered herein.

5.1.3.2 FDP_SDI.2 Stored Data Integrity Monitoring and Action

FDP_SDI.2.1 The TSF shall monitor user data stored within the TSC for **[*integrity errors*]** on all objects, based on the following attributes: **[*D.APP_CODE, D.APP_I_DATA, D.PIN, D.APP_KEYs*]**.

FDP_SDI.2.2 Upon detection of a data integrity error, the TSF shall **[maintain a secure state and return an error message]**.

5.1.3.3 FPT_RVM.1 Reference Mediation

FPT_RVM.1.1 The TSF shall ensure that TSP enforcement functions are invoked and succeed before each function within the TSC is allowed to proceed.

> Application note: Execution of native code is not within the TSC. Nevertheless, access to native methods from the Java Card System is subject to TSF control, as there is no difference in the interface or the invocation mechanism between native and interpreted methods.

5.1.3.4 FPT_FLS.1 Failure with Preservation of Secure State

FPT_FLS.1.1/JCS The TSF shall preserve a secure state when the following types of failures occur: ***those associated to the potential security violations described in FAU_ARP.1/JCS.***

> Application note: The *JCRE Context* is the *Current Context* when the VM begins running after a card reset ([JCRE221], §6.2.3). Behavior of the TOE on power loss and reset is described in [JCRE221], §3.5, and §7.1.

5.1.3.5 FPR_UNO.1 Unobservability

FPR_UNO.1.1 The TSF shall ensure that **[subjects S.Package]** are unable to observe the operation **[all operations]** on **[secret keys and PIN codes]** by **[other subjects S.Package]**.

> Application note: Although it is not required in [JCRE221] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.

5.1.3.6 FPT_TST.1 TSF Testing

FPT_TST.1.1 The TSF shall run a suite of self-tests ***during initial start-up (at each power on)*** to demonstrate the correct operation of ***the TSF***[26].

[26] Editorial change according to final interpretation FI056 where “the TSF” is part of a selection operation.

Application note: TSF-testing is not mandatory in [JCRE221], but appears in most of security requirements documents for masked applications. Testing could also occur randomly.

FPT_TST.1.2 The TSF shall provide authorized users with the capability to verify the integrity of ***TSF data***[27].

FPT_TST.1.3 The TSF shall provide authorized users with the capability to verify the integrity of stored TSF executable code.

5.1.4 AID Management

5.1.4.1 FMT_MTD.1 Management of TSF Data

(See ***FMT_SMR.1.1/JCRE*** for the roles)

FMT_MTD.1.1/JCRE The TSF shall restrict the ability to **modify** the **list of registered applets'** *AID* to **the JCRE [only]**.

Application note: The *installer* and the *JCRE* manage some other TSF data such as the *applet* life cycle or *CAP file*s, but this management is implementation specific. Objects in the Java programming language may also try to query *AIDs* of installed *applet*s through the **lookupAID(…)** API method.

Application note: The *installer, applet deletion manager* or even the card manager may be granted the right to modify the list of registered applets' *AIDs* in specific implementations (possibly needed for installation and deletion; see #.**DELETION** and #.**INSTALL**).

5.1.4.2 FMT_MTD.3 Secure TSF data

FMT_MTD.3.1 The TSF shall ensure that only secure values are accepted for TSF data.

5.1.4.3 FIA_ATD.1 User Attribute Definition

FIA_ATD.1.1/AID The TSF shall maintain the following list of security attributes belonging to individual users: ***the AID and version number of each package, the AID of each registered applet, and whether a registered applet is currently selected for execution ([JCVM221], §6.5).***

5.1.4.4 FIA_UID.2 User Identification before any Action

FIA_UID.2.1/AID The TSF shall require each user to identify itself before allowing any other TSF-mediated actions on behalf of that user.

Application note: By users here it must be understood the ones associated to the *packages (or applets)* which act as subjects of policies. In the Java Card System, every action is always performed by an identified user interpreted here as the currently selected *applet* or the *package* that is the *subject*'s owner. Means of identification are provided during the loading procedure of the *package* and the registration of *applet* instances.

[27] Editorial change according to final interpretation FI056 where "the TSF data" is part of a selection operation.

Application note: The role *JCRE* defined in *FMT_SMR.1/JCRE* is attached to an IT security function rather than to a "user" of the CC terminology. The *JCRE* does not "identify" itself with respect to the TOE, but it is a part of it.

5.1.4.5 FIA_USB.1 User-Subject binding

Note: According to final interpretation #137 the SFR FIA_USB.1 is rewritten as:

FIA_USB.1.1 *The TSF shall associate the following user security attributes with subjects acting on the behalf of that user:* ***[active Context and SELECTed applet Context security attribute].***

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:* ***[rules defined in FDP_ACF.1.1/FIREWALL, FMT_MSA.2.1/JCRE, and FMT_MSA.3.1/FIREWALL and corresponding application notes]****.*

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:* ***[rules defined in FMT_MSA.1.1/JCRE].***

Application note: For *S.PACKAGE*s, the Context security attribute plays the role of the appropriate user security attribute; see ***FMT_MSA.1.1/JCRE*** above.

5.1.5 SCPG Security Functional Requirements

For this evaluation the smart card platform belongs to the TOE and the functional requirements are stated here as functional requirements for the TOE.

5.1.5.1 FPT_AMT.1 Abstract Machine Testing

FPT_AMT.1.1/SCP The TSF shall run a suite of tests ***during initial start-up (at each power on)*** to demonstrate the correct operation of the security assumptions provided by the abstract machine that underlies the TSF.

5.1.5.2 FPT_FLS.1 Failure with preservation of a Secure State

This assignment operation of the functional requirement has been taken over from the ST of the certified hardware platform NXP P5CT072V0P that is conformant to [PP0002].

FPT_FLS.1.1/SCP The TSF shall preserve a secure state when the following types of failures occur: ***[exposure to operating conditions which may not be tolerated according to the requirement Limited fault tolerance (FRU_FLT.2) and where therefore a malfunction could occur and failures detected by TSF according to FPT_TST.1]***.

5.1.5.3 FRU_FLT.2 Limited Fault Tolerance

The functional requirement FRU_FLT.2 is hierarchical to the requirement FRU_FLT.1 that is included in [JCSPP] and therefore includes this requirement. It has been taken over from the ST of the certified hardware platform NXP P5CT072V0P that is conformant to [PP0002].

FRU_FLT.2.1/SCP The TSF shall ensure the operation of all the TOE[28] capabilities when the following failures occur: ***[exposure to operating conditions which may not be tolerated according to the requirement Failure with preservation of a secure state (FPT_FLS.1)]***.

REFINEMENT: ***The term "failure" above means "circum stances". The TOE prevents failures for the "circumstances" defined above.***

These components shall be used to specify the list of *SCP* capabilities supporting the Java Card System/CM that will still be operational at the occurrence of the mentioned failures (EEPROM worn out, lack of EEPROM, random generator failure).

5.1.5.4 FPT_PHP.3 Resistance to Physical Attack

This functional requirement has been taken over from the ST of the certified hardware platform NXP P5CT072V0P that is conformant to [PP0002].

FPT_PHP.3.1/SCP The TSF shall resist **[physical manipulation and physical probing]** to the **[TSF]** by responding automatically such that the TSP is not violated.

REFINEMENT: ***The TOE will implement appropriate measures to continuously counter physical manipulation and physical probing. Due to the nature of these attacks (especially manipulation) the TOE can by no means detect attacks on all of its elements. Therefore, permanent protection against these attacks is required ensuring that the TSP could not be violated at any time. Hence, "automatic response" means here (i) assuming that there might be an attack at any time and (ii) countermeasures are provided at any time.***

5.1.5.5 FPT_SEP.1 TSF Domain Separation

FPT_SEP.1.1/SCP The TSF shall maintain a security domain for its own execution that protects it from interference and tampering by untrusted subjects.

FPT_SEP.1.2/SCP The TSF shall enforce separation between the security domains of subjects in the TSC.

REFINEMENT: ***Those parts of the TOE which support the security functional requirements "Limited fault tolerance (FRU_FLT.2)" and "Failure with preservation of secure state (FPT_FLS.1)" shall be protected from interference of the Smartcard Embedded Software.***

5.1.5.6 FPT_RCV Trusted Recovery

FPT_RCV.3.1/SCP When automated recovery from ***[detected integrity errors in D.APP_CODE, D.APP_I_DATA, D.PIN, D.APP_KEYs]*** is not possible, the TSF shall enter a maintenance mode where the ability to return the TOE to a secure state is provided.

FPT_RCV.3.2/SCP For **[power failure]**, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

[28] Editorial Change: The word "TOE's" was replaced by "TOE".

FPT_RCV.3.3/SCP The functions provided by the TSF to recover from failure or service discontinuity shall ensure that the secure initial state is restored without exceeding **[100%]** for loss of TSF data or objects within the TSC.

FPT_RCV.3.4/SCP The TSF shall provide the capability to determine the objects that were or were not capable of being recovered.

FPT_RCV.4.1/SCP The TSF shall ensure that reading from and writing to static and objects' fields interrupted by power loss have the property that the SF either completes successfully, or for the indicated failure scenarios, recovers to a consistent and secure state.

Application note: This requirement comes from the specification of the Java Card platform but is obviously supported in the implementation by a low-level mechanism.

Application note: In case of detected integrity errors in D.APP_CODE, D.APP_I_DATA, D.PIN, D.APP_KEYs, the card should stop and wait for the maintenance action reset. When the card is reset, the hardware should be re-initialized with the ability to return the TOE to a secure state (FPT_RCV.3.1/SCP).
A power failure should cause a reset and after a reset a secure state must be entered (FPT_RCV.3.2/SCP).
A transaction mechanism should ensure that no TSF data or objects are lost when a secure state is restored. Therefore, the TOE must check in advance if enough space is left in the transaction buffer (FPT_RCV.3.3/SCP, FPT_RCV.4.1/SCP).
The capability to determine the objects that were or were not capable of being recovered (FPT_RCV.3.4/SCP) should be given by the kind of memory used for the objects

5.1.5.7 FPT_RVM.1 Reference Mediation

FPT_RVM.1.1/SCP The TSF shall ensure that the TOE enforcement functions (TSP) are invoked and succeed before each function within the TSC is allowed to proceed.

Application note: This component supports *O.*SCP.*SUPPORT*, which in turn contributes to the secure operation of the TOE, by ensuring that these latter and supporting platform security mechanisms cannot be bypassed.

5.1.6 *CMGRG* Security Functional Requirements

This group contains the security requirements for the card manager. For this evaluation the card manager belongs to the TOE and the functional requirements are stated here as functional requirements for the TOE.

5.1.6.1 FDP_ACC.1 Subset Access Control

FDP_ACC.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** on **[subjects: S.PACKAGE(CM), S.PACKAGE, S.JCRE; objects: D.App_Code, and all operations among subjects and objects covered by the *SFP*]**.

5.1.6.2 FDP_ACF.1 Security Attribute based Access Control

FDP_ACF.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to objects based on **[the security attributes of S.PACKAGE(CM): Card Life Cycle State as defined in [GP] section 5.1: *OP_READY, INITIALIZED, SECURED, CARD_LOCKED, TERMINATED*]**.

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

3. **Loading of D.App_Code must only be performed by S.PACKAGE(CM) in card life cycle states OP_READY, INITIALIZED or SECURED and must not be performed in card life cycle states CARD_LOCKED or TERMINATED**

4. **Loading of D.App_Code must only be performed S.PACKAGE(CM) after initiation of a Secure Channel.**

5. **S.PACKAGE(CM) is allowed to set the Card Life Cycle States *OP_READY, INITIALIZED, SECURED, CARD_LOCKED,* and *TERMINATED.*]**.

FDP_ACF.1.3/CMGR The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: **[none]**.

FDP_ACF.1.4/CMGR The TSF shall explicitly deny access of subjects to objects based on the **[following rules:]**.

1. **If the card life cycle state is TERMINATED, the TOE is blocked, and the access of subjects is no more allowed.**

5.1.6.3 FMT_MSA.1 Management of Security Attributes

FMT_MSA.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to restrict the ability to **[modify]** the security attributes **[card life cycle state]** to **[S.PACKAGE(CM)]**.

5.1.6.4 FMT_MSA.3 Static Attribute Initialization

FMT_MSA.3.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to provide ***restrictive*** default values for security attributes that are used to enforce the *SF*P.

FMT_MSA.3.2/CMGR The TSF shall allow the **[no roles]** to specify alternative initial values to override the default values when an object or information is created.

5.1.6.5 FMT_SMR.1 Security Roles

FMT_SMR.1.1/CMGR The TSF shall maintain the roles: **[S.PACKAGE(CM)]**.

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

5.1.6.6 FIA_UID.1 Timing of Identification

FIA_UID.1.1/CMGR The TSF shall allow **[execution of S.PACKAGE]** on behalf of the user to be performed before the user is identified.

FIA_UID.1.2/CMGR The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

5.1.7 Further Functional Requirements not contained in [JCSPP]

The SFR in this section are not contained in [JCSPP]. SFR defined in sections 5.1.7.1 - 5.1.7.15 are specifically related to security functionality of the Card Manager.

5.1.7.1 FDP_ETC.1 Export of User Data without Security Attributes

FDP_ETC.1.1 The TSF shall enforce the **[*CARD CONTENT MANAGEMENT access control SFP*]** when exporting user data, controlled under the SFP(s), outside of the TSC.

FDP_ETC.1.2 The TSF shall export the user data without the user data's associated security attributes.

5.1.7.2 FDP_ITC.1 Import of User Data without Security Attributes

FDP_ITC.1.1 The TSF shall enforce the **[*CARD CONTENT MANAGEMENT access control SFP*]** when importing user data, controlled under the SFP, from outside of the TSC.

Application note User data are: D.APP_CODE, D.PIN, D.APP_C_DATA, D.APP_I_DATA, D.APP_KEYs. The most common importation of user data is normally package loading and applet installation on the behalf of the installer. In the case of this ST, loading is handled separately during manufacturing and not through the card manager loader. Security attributes consist of the shareable flag of the class component, AID and version numbers of the package, maximal operand stack size and number of local variables for each method, and export and import components. Other instances of importing user data include setting the pin and key import.

FDP_ITC.1.2 The TSF shall ignore any security attributes associated with the user data when imported from outside the TSC.

FDP_ITC.1.3 The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TSC: **[none]**.

5.1.7.3 FIA_AFL.1 Basic authentication Failure Handling

FIA_AFL.1.1/PIN The TSF shall detect when ***[an administrator configurable positive integer within [1 and 127]]*** unsuccessful authentication attempts occur related to ***[any user authentication using D.PIN]***.

FIA_AFL.1.2/PIN When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall ***[block the authentication with D.PIN]***.

FIA_AFL.1.1/CMGR The TSF shall detect when ***[10 consecutive]*** unsuccessful authentication attempts occur related to ***[any user authentication to the CARDMANAGER (S.PACKAGE(CM) via Secure Messaging using D.APP_KEYs]***.

FIA_AFL.1.2/CMGR When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall ***[block the CARD]***.

5.1.7.4 FIA_UAU.1 Timing of Authentication

FIA_UAU.1.1 The TSF shall allow **[the following TSF mediated command]** on behalf of the user to be performed before the user is authenticated.

Command	Objects
Get Data	ISD DATA [ISSUER IDENTIFICATION NUMBER], ISD DATA [CARD IMAGE NUMBER], PLATFORM DATA [CARD RECOGNITION DATA], ISD DATA [KEY INFORMATION TEMPLATE], ISD DATA [SCP INFORMATION], PLATFORM DATA [MANUFACTURING]
Select Applet	
Initialize Update	APDU BUFFER
External Authenticate	APDU BUFFER

FIA_UAU.1.2 The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

5.1.7.5 FIA_UAU.3 Unforgeable Authentication

FIA_UAU.3.1/CMGR The TSF shall **[prevent]** use of authentication data that has been forged by any user of the TSF.

FIA_UAU.3.2/CMGR The TSF shall **[prevent]** use of authentication data that has been copied from any other user of the TSF.

Note: Only applicable for card manager authentication and not for authentication with D.PIN.

5.1.7.6 FIA_UAU.4 Single-use Authentication Mechanisms

FIA_UAU.4.1/CMGR The TSF shall prevent reuse of authentication data related to **[Card Manager authentication mechanism]**.

5.1.7.7 FTP_ITC.1 Inter-TSF Trusted Channel – none

FTP_ITC.1.1/CMGR The TSF shall provide a communication channel between itself and a remote trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure.

FTP_ITC.1.2/CMGR The TSF shall permit **[the remote trusted IT product]** to initiate communication via the trusted channel.

FTP_ITC.1.3/CMGR The TSF shall initiate communication via the trusted channel for **[loading of D.App_Code, setting the Card Life Cycle State]**.

5.1.7.8 FAU_SAA.1 Potential Violation Analysis

FAU_SAA.1.1 The TSF shall be able to apply a set of rules in monitoring the audited events and based upon these rules indicate a potential violation of the TSP.

FAU_SAA.1.2 The TSF shall enforce the following rules for monitoring audited events:

a) Accumulation or combination of **[*the following auditable events*]** known to indicate a potential security violation;

List of auditable events:

1. ***Abnormal environmental conditions (frequency, voltage, temperature)***
2. ***Physical tampering***
3. ***EEPROM failure audited through exceptions in the read/write operations and inconsistency check;***
4. ***Card Manager life cycle state inconsistency audited through the life cycle checks in all administrative operations and the self test mechanism on start-up.***
5. **Applet life cycle inconsistency.**
6. ***Corruption of check-summed objects.***
7. ***Illegal access to the previously defined D.JAVA_OBJECT objects audited through the firewall mechanism.***
8. ***Unavailability of resources audited through the object allocation mechanism.***

9. ***Abortion of a transaction in an unexpected context (see abortTransaction(), [JCAPI221] and ([JCRE222], §7.6.2)***
10. ***Violation of the Firewall or JCVM SFPs.***
11. ***Array overflow***
12. ***Other runtime errors related to applet's failure (like uncaught exceptions)***
13. ***Card tearing (unexpected removal of the Card out of the CAD) and power failure***

b) **[*no other rules*]**.

Application Note: Off-card entities are provided with the basic ability to find out certain pieces of information about the card through the Open Platform GET DATA command. Authenticated off-card entities can determine other information such as the AIDs of on-card Applications and Life Cycle states using other APDU commands. This provides a limited ability to "audit" the card, and is probably sufficient for most purposes.

5.1.7.9 FMT_SMF.1 Specification of Management Function

FMT_SMF.1.1 The TSF shall be capable of performing the following security management functions: **[*modify the behavior of functions, modify the active context and the SELECTed applet Context, modify the list of registered applets' AID, modify the card life cycle state attribute*]**.

5.1.7.10 FCS_RND.1 Quality metric for Random Numbers

FCS_RND.1.1 The TSF shall provide a mechanism to generate random numbers that meet the ***[class K3 of [AIS 20] with SOF-high].***

5.1.7.11 FPT_EMSEC.1 TOE Emanation

FPT_EMSEC.1.1 The TOE shall not emit ***[variations in power consumption or timing during command execution]*** in excess of ***[non-useful information]*** enabling access to ***[TSF data: D.JCS_KEYs and D.CRYPTO]*** and ***[User data: D.PIN, D.APP_KEYs]***.

FPT_EMSEC.1.2 The TSF shall ensure ***[that unauthorized users]*** are unable to use the following interface ***[electrical contacts]*** to gain access to ***[TSF data: D.JCS_KEYs and D.CRYPTO]*** and ***[User data: D.PIN, D.APP_KEYs]***.

5.1.7.12 FMT_LIM.1 Limited Capabilities

FMT_LIM.1.1 The TSF shall be designed in a manner that limits their capabilities so that in conjunction with "Limited availability

(FMT_LIM.2)" the following policy is enforced ***[Deploying Test Features after TOE Delivery does not allow***

1. **User Data to be disclosed or manipulated**
2. **TSF data to be disclosed or manipulated**
3. **software to be reconstructed and**
4. **substantial information about construction of TSF to be gathered which may enable other attacks].**

5.1.7.13 FMT_LIM.2 Limited Availability

FMT_LIM.2.1 The TSF shall be designed in a manner that limits their availability so that in conjunction with "Limited capabilities (FMT_LIM.1)" the following policy is enforced **[Deploying Test Features after TOE Delivery does not allow**

1. **User Data to be disclosed or manipulated**
2. **TSF data to be disclosed or manipulated**
3. **software to be reconstructed and**
4. **substantial information about construction of TSF to be gathered which may enable other attacks].**

5.1.7.14 FPT_PHP.1 Passive Detection of physical Attack

FPT_PHP.1.1 The TSF shall provide unambiguous detection of physical tampering that might compromise the TSF.

FPT_PHP.1.2 The TSF shall provide the capability to determine whether physical tampering with the TSF's devices or TSF's elements has occurred.

5.1.7.15 FPT_TDC.1 Inter-TSF basic TSF Data Consistency

FPT_TDC.1.1 The TSF shall provide the capability to consistently interpret the CAP files (shared between the card manager and the TOE), the bytecode and its data arguments (shared with applets and API packages), when shared between the TSF and another trusted IT product.

FPT_TDC.1.2 The TSF shall use the following rules when interpreting the TSF data from another trusted IT product:

- The [JCVM221] specification;
- Reference export files;
- The ISO 7816-6 rules;
- The EMV specification

5.2 TOE Security Assurance Requirements

The assurance requirements of the Security Target are **EAL4** augmented by **ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3**.

The assurance requirements ensure, among others, the security of the TOE during its development and production. We present here some application notes on the assurance requirements included in the EAL of the ST. *These are not to be considered as iteration or refinement of the original components.*

- ***ACM_AUT.1*** Partial Configuration Management automation
- ***ACM_CAP.4*** Generation support and acceptance procedures
- ***ACM_SCP.2*** Problem tracking Configuration Management coverage

These components contribute to the integrity and correctness of the TOE during its development. Procedures dealing with physical, personnel, organizational, technical measures for the confidentiality and integrity of *Java Card System* software (source code and any associated documents) shall exist and be applied in software development.

- ***ADV_FSP.2*** Fully defined external interfaces
- ***ADV_HLD.2*** Security enforcing high-level design
- ***ADV_LLD.1*** Descriptive low-level design
- ***ADV_RCR.1*** Informal correspondence demonstration
- ***ADV_SPM.1*** Informal TOE security policy model

These SARs ensure that the TOE will be able to meet its security requirements and fulfill its objectives. The *Java Card System* shall implement the [JCAPI221]. The implementation of the Java Card API shall be designed in a secure manner, including specific techniques to render sensitive operations resistant to state-of-art attacks.

- ***ADO_DEL.2*** Detection of modification

This SAR ensures the integrity of the TOE and its documentation during the transfer of the TOE between all the actors appearing in the first two stages. Procedures shall ensure protection of TOE material/information under delivery and storage that corrective actions are taken in case of improper operation in the delivery process and storage and that people dealing with the procedure for delivery have the required skills.

- ***ADO_IGS.1*** Installation, generation, and start-up procedures
- ***AGD_ADM.1*** Administrator guidance
- ***AGD_USR.1*** User guidance

These SARs ensure proper installation and configuration: the TOE will be correctly configured and the TSFs will be put in good working order. The administrator is the card issuer, the platform developer, the card embedder or any actor who participates in the fabrication of the TOE once its design and development is complete (its source code is available and released by the TOE designer). The users are applet developers, the card manager developers, and possibly the final user of the TOE.

The *applet* and API *package*s programmers should have a complete understanding of the concepts defined in [JCRE] and [JCVM]. They must delegate key management, PIN management and cryptographic operations to dedicated APIs. They should carefully

consider the effect of any possible exception or specific event and take appropriate measures (such as catch the exception, abort the current transaction, and so on.). They must comply with all the recommendations given in the platform programming guide as well. Failure to do so may jeopardize parts of (or even the whole) *applet* and its confidential data.

This guidance also includes the fact that sharing object(s) or data between *applet*s (through *shareable interface* mechanism, for instance) must include some kind of authentication of the involved parties, even when no sensitive information seems at stake (so-called “defensive development”).

- ***ALC_LCD.1*** Developer defined life-cycle model
- ***ALC_TAT.1*** Well-defined development tools

It is assumed that security procedures are used during all manufacturing and test operations through the production phase to maintain confidentiality and integrity of the TOE and of its manufacturing and test data (to prevent any possible copy, modification, retention, theft or unauthorized use).

- ***ATE_COV.2*** Analysis of Coverage
- ***ATE_DPT.1*** Testing: high-level design
- ***ATE_FUN.1*** Functional testing
- ***ATE_IND.2*** Independent testing - sample

The purpose of these SARs is to ensure whether the TOE behaves as specified in the design documentation and in accordance with the TOE security functional requirements. This is accomplished by determining that the developer has tested the security functions against its functional specification and high level design, gaining confidence in those tests results by performing a sample of the developer’s tests, and by independently testing a subset of the security functions.

- ***AVA_SOF.1*** Strength of TOE security function evaluation

The objectives of this SARs are to review the identified vulnerabilities and to determine whether SOF claims are made in the ST for all non-cryptographic, probabilistic or permutational mechanisms.

Augmentation of level EAL4 results from the selection of the following four SARs:

- ***ADV_IMP.2*** Implementation of the TSF.

EAL4 requires through imposition of ***ADV_IMP.1*** the description of a subset of the implementation representation in order to capture the detailed internal working of the TSF. The component ***ADV_IMP.2*** requires the developer to provide the implementation representation for the entire TSF.

- ***ALC_DVS.2*** Sufficiency of security measures

EAL4 requires for the development security the assurance component ***ALC_DVS.1***. This dictates a documentation and check of the security measures in the development environment. The component ***ALC_DVS.2*** requires additionally a justification, that the measures provide the necessary level of protection.

- ***AVA_VLA.4*** Highly resistant

EAL4 requires for the vulnerability assessment the assurance component ***AVA_VLA.2***. Its aim is to determine whether the TOE, in its intended environment, has vulnerabilities exploitable by attackers with low attack potential. In order to provide the necessary level of protection, EAL4 is augmented with the component ***AVA_VLA.4***, which requires that the TOE is resistant against attackers with high attack potential.

- ***AVA_MSU.3*** Analysis and testing for insecure states

EAL4 requires for the misuse analysis the assurance component ***AVA_MSU.2***. This requires the developer to provide guidance documentation and documentation of the analysis of the guidance documentation. The component ***AVA_MSU.3*** further requires to validate and to confirm this analysis through testing by an evaluator.

5.2.1 Minimum Strength of Function (SOF) Claim

The minimum level of strength of the security functions that are fulfilling these security requirements is to be **SOF-high**.

5.3 Security Requirements for the IT environment

5.3.1 BCVG Security Functional Requirements

This group of requirements concerns bytecode verification. A bytecode verifier can be understood as a process that acts as a filter on a *CAP* file verifying that the bytecodes of the methods defined in the file conform to certain well-formed requirements. As mentioned in §2.3.1, there are different techniques that have been proposed for performing those checks. The solution described in [JCBV], for example, is based on a data flow analysis and makes use of an abstract interpreter. The abstract interpreter simulates execution of each instruction, using types of the data being operated on instead of values. For each instruction, the state of the operand stack and local variables are compared to the type(s) required during execution, and then are updated according to the operation of the instruction.
The main component of this group of functional requirements is an information flow control policy, which describes the constraints imposed on the operations (the bytecodes) that make information flow between the subjects (local variables, operand stack, fields).
The group is composed of three sub-groups. The first one constitutes a complete information flow control policy with hierarchical attributes, which describes the type constraints imposed on the bytecodes. That typing policy strongly depends on having a secure configuration of the attributes it is based on. Such secure configurations are strongly related to the constraints imposed on the structure of the *CAP* file format by Sun specifications, and constitute a second important sub-group of requirements. Finally, the third sub-group requires bytecode verification to prevent any operand stack overflow that could arrive during the interpretation of bytecodes.

5.3.1.1 FDP_IFC.2 Complete Information Flow Control

FDP_IFC.2.1/BCV	The TSF shall enforce the ***TYPING information flow control SFP*** on ***S.LOCVAR, S.STCKPOS, S.FLD, S.MTHD*** and all operations that cause that information to flow to and from subjects covered by the SFP.

Subjects[29] (prefixed with an "*S*") covered by this policy are:

Subject	Description
S.LOCVAR	Any local variable of the currently executed method.
S.STCKPOS	Any operand stack position of the currently executed method.
S.FLD	Any field declared in a package loaded on the card.
S.MTHD	Any method declared in a package loaded on the card.

The operations (prefixed with "*OP*") that make information flow between the subjects are all bytecodes. For instance, the *aload_0* bytecode causes information to flow from the local variable 0 to the top of the operand stack; the bytecode *putfield(x)* makes information flow from the top of the operand stack to the field x; and the *return a* bytecode makes information flow out of the currently executed method.

Operation	Description
OP.BYTECODE(BYTCD)	Any bytecode for the Java Card platform ("Java Card bytecode").

The information (prefixed with an "**I**") controlled by the typing policy are the bytes, shorts, integers, references and return addresses contained in the different storage units of the JCVM (local variables, operand stack, static fields, instance fields and array positions).

Information	Description
I.BYTE(BY)	Any piece of information that can be encoded in a byte.
I.SHORT(SH)	Any piece of information that can be encoded in a short value.
I.INT(W1,W2)	Any piece of information that can be encoded in an integer value, which in turn is encoded in two words w1 and w2.
I.REFERENCE(RF)	Any reference to a class instance or an array.
I.ADDRESS(ADRS)	Any return address of a subroutine.

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

5.3.1.2 FDP_IFF.2 Hierarchical Security Attributes

See FMT_MSA.1 for more information about security attributes.

FDP_IFF.2.1/BCV The TSF shall enforce the ***TYPING information flow control SFP*** based on the following types of subject and information security attributes: ***(1) type attribute of the information, (2) type attribute of***

[29] Information flow policies control the flow of information between "subjects". This is a purely terminological choice; those "subjects" can merely be passive containers. They are not to be confused with the "active entities" of access control policies.

the storage units of the JCVM, (3) class attribute of the fields and methods, (4) bounds attribute of the methods.

The following table describes which security attributes are attached to which subject/information of our policy.

Subject/Information	Attributes
S.LOCVAR	*TYPE*
S.STCKPOS	*TYPE*
S.FLD	*TYPE, CLASS*
S.MTHD	*TYPE, CLASS, BOUNDS*
I.BYTE(BY)	*TYPE*
I.SHORT(SH)	*TYPE*
I.INT(W1,W2)	*TYPE*
I.REFERENCE(RF)	*TYPE*
I.ADDRESS(ADRS)	*TYPE*

The following table describes the security attributes.

Attribute Name	Description
TYPE	Either the type attached to the information, or the type held or declared by the subject.
CLASS	The class where a field or method is declared.
BOUNDS	The start and end of the method code inside the method component of the CAP file where it is declared.

The *TYPE* security attribute attached to local variables and operand stack positions is the type of information they currently hold. The *TYPE* attribute of the fields and the methods is the type declared for them by the programmer.

The *BOUNDS* attribute of a method is used to prevent control flow to jump outside the currently executed method.

The following table describes the possible values for each security attribute.

Name	Description
TYPE	byte, short, int_1, int_2, any class name *C*, *T*[] with *T* any type in the Java Card platform (“Java Card type”), T_0 (T_1 x_1, …. T_n x_n) with T_0,.. T_n any Java Card type, RetAddrs(adrs), Top, Null, ⊥**.**
CLASS	The name of a class, represented as a reference into the class Component of one of the packages loaded on the card.
BOUNDS	Two integers marking a rank into the method component of a package loaded on the card.

Byte values have type ***byte*** and short values have type ***short***. The first and second halves of an integer value have respectively type int_1, and int_2. The type of a reference to an instance of the class *C* is ***C*** itself***.*** A reference to an array of elements of type *T* has type ***T[]***. From the previous basic types it is possible to build the type T_0 (T_1 x_1, …. T_n x_n) of a method. A return address *adrs* of a subroutine has type ***RetAddrss(****adrs****)***. Finally, the former Java Card types are extended with three extra types **Top**, **Null** and ⊥**,** so that the domain of types forms a complete lattice. **Top** is the type of any piece of data, that is, the maximum of the lattice. **Null** is the type of the default value null of all the reference types (classes and arrays). ⊥ is the type of an element that belongs to all types (for instance the value 0, provided that null is represented as zero).

FDP_IFF.2.2/BCV The TSF shall permit an information flow between a controlled subject and controlled information *through*[30] a controlled operation if the following rules, based on the ordering relationships between security attributes, hold:

The following rules constitute a synthetic formulation of the information flow control:

R.JAVA.6 If the bytecode pushes values from the operand stack, then there are a sufficient number of values on the stack and the values of the attribute *TYPE* of the top positions of the stack is appropriate with respect to the ones expected by the bytecode.

R.JAVA.7 If the bytecode pushes values onto the operand stack, then there is sufficient room on the operand stack for the new values. The values, with the appropriate attribute *TYPE* value are added to the top of the operand stack.

R.JAVA.8 If the bytecode modifies a local variable with a value with attribute *TYPE* T, it must be recorded that the local variable now contains a value of that type. In addition, the variable shall be among the local variables of the method.

R.JAVA.9 If the bytecode reads a local variable, it must be ensured that the specified local variable contains a value with the attribute *TYPE* specified by the bytecode.

R.JAVA.10 If the bytecode uses a field, it must be ensured that its value is of an appropriate type. This type is indicated by the *CLASS* attribute of the field.

R.JAVA.11 If the bytecode modifies a field, then it must be ensured that the value to be assigned is of an appropriate type. This type is indicated by the *CLASS* attribute of the field

R.JAVA.12 If the bytecode is a method invocation, it must be ensured that it is invoked with arguments of the appropriate type. These types are indicated by the *TYPE* and *CLASS* attributes of the method.

[30] Editorial change: The word “via” was replaced by “through” as proposed in [JCSPP].

R.JAVA.13 If the bytecode is a branching instruction, then the bytecode target must be defined within the *BOUNDS* of the method in which the branching instruction is defined.

Application note: The rules described above are strongly inspired in the rules described in section 4.9 of [JVM], ***Second Edition***. The complete set of typing rules can be derived from the "Must" clauses from Chapter 7 of [JCVM221] as instances of the rules defined above.

FDP_IFF.2.3/BCV The TSF shall enforce the ***following additional information flow control SFP rules: none.***

FDP_IFF.2.4/BCV The TSF shall provide the following list of additional SFP capabilities: ***none.***

FDP_IFF.2.5/BCV The TSF shall explicitly authorize an information flow based on the following rules: ***none***.

FDP_IFF.2.6/BCV The TSF shall explicitly deny an information flow based on the following rules: ***none.***

FDP_IFF.2.7/BCV The TSF shall enforce the following relationships for any two valid information flow control security attributes:

a) There exists an ordering function that, given two valid security attributes, determines if the security attributes are equal, if one security attribute is greater than the other, or if the security attributes are incomparable; and

b) There exists a least upper bound in the set of security attributes, such that, given any two valid security attributes, there is a valid security attribute that is greater than or equal to the two valid security attributes; and

c) There exists a greatest lower bound in the set of security attributes, such that, given any two valid security attributes, there is a valid security attribute that is not greater than the two valid security attributes.

Application note: The order relationship between Java Card types is described, for instance, in the description of the **checkcast** bytecode of [JCVM221]. That relation is with the following rules:

- **Top** is the maximum of all types;
- **Null** is the minimum of all classes and array types;
- ⊥ is the minimum of all types.

These three extra types are introduced in order to satisfy the two last items in requirement FDP_IFF.2.7.

5.3.1.3 FMT_MSA.1 Management of Security Attributes

(See ***FMT_SMR.1.1/BCV*** (p. 86) for the roles)

FMT_MSA.1.1/BCV.1 The TSF shall enforce the ***TYPING information flow control SFP*** to restrict the ability to **modify** the ***TYPE security attribute of the fields and methods***[31] to **none.**

FMT_MSA.1.1/BCV.2 The TSF shall enforce the ***TYPING information flow control SFP*** to restrict the ability to **modify** the ***TYPE security attribute of local variables and operand stack position*** to **the role Bytecode Verifier.**

Application note: The TYPE attribute of the local variables and the operand stack positions is identified to the attribute of the information they hold. Therefore, this security attribute is possibly modified as information flows. For instance, the rules of the typing function enable information to flow from a local variable *lv* to the operand stack by the operation *sload*, provided that the value of the type attribute of *lv* is ***short***. This operation hence modifies the type attribute of the top of the stack. The modification of the security attributes should be done according to the typing rules derived from ***Chapter 7 of [JCVM221]***.

5.3.1.4 FMT_MSA.2 Secure Security Attributes

FMT_MSA.2.1/BCV The TSF shall ensure that only secure values are accepted for security attributes.

Application note: During the type verification of a method, the bytecode verifier makes intensive use of the information provided in the CAP format like the sub-class relationship between the classes declared in the package, the type and class declared for each method and field, the rank of exceptions associated to each method, and so on. All that information can be thought of as security attributes used by the bytecode verifier, or as information relating security attributes. Moreover, the bytecode verifier relies on several properties about the CAP format. All the properties on the CAP format required by the bytecode verifier could, for instance, be completely described in the TSP model, and the bytecode verifier should ensure that they are satisfied before starting type verifications. Examples of such properties are:

- Correspondences between the different components of the CAP file (for instance, each class in the class component has an entry in the descriptor component).
- Pointer soundness (example: the index argument in a static method invocation always has an entry in the constant pool);
- Absence of hanged pointers (example: each exception handler points to the beginning of some bytecode);
- Redundant information (enabling different ways of searching for it);
- Conformance to the Java Language Specification respecting the access control features mentioned in §2.2 of [JCVM221].
- Packages that are loaded post-issuance can not contain native code.

[31] Editorial refinement as already proposed in [JCSPP].

5.3.1.5 FMT_MSA.3 Static Attribute Initialization

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

Application note: The TYPE attribute of the fields and methods is fixed by the application provider and never modified. When a method is invoked, the operand (type) stack is empty. The initial type assigned to those local variables that correspond to the method parameters is the type the application provider declared for those parameters. Any other local variable used in the method is set to the default value Top.

FMT_MSA.3.2/BCV The TSF shall allow ***the following role(s)*** to specify alternative initial values to override the default values when an object or information is created: *none*[32].

Application note: The intent is to have none of the identified roles to have privileges with regards to the default values of the TYPE attributes.

5.3.1.6 FMT_SMR.1 Security Roles

FMT_SMR.1.1/BCV The TSF shall maintain the roles: ***Bytecode Verifier.***

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

5.3.1.7 FMT_SMF.1 Specification of Management Functions

FMT_SMF.1.1/BCV The TSF shall be capable of performing the following security management functions: ***[Modify the TYPE security attribute of local variables and operand stack position].***

Please note that this SFR has been added according to final interpretation 065 which is already included in [CC].

5.3.1.8 FRU_RSA.1 Maximum Quotas

FRU_RSA.1.1/BCV The TSF shall enforce maximum quotas of the following resources: **the operand stack** and **the local variables** that **a method**[33] can use **simultaneously.**

5.3.2 Trusted Channel

5.3.2.1 FTP_ITC.1 Inter-TSF trusted Channel – none

FTP_ITC.1.1/ENV The TSF shall provide a communication channel between itself and a remote trusted IT product that is logically distinct from other communication channels and provides assured

[32] Editorial Change: The sentence was changed for better readability.

[33] Editorial refinement as already proposed in [JCSPP].

	identification of its end points and protection of the channel data from modification or disclosure.
FTP_ITC.1.2/ENV	The TSF shall permit **[the TSF]** to initiate communication via the trusted channel.
FTP_ITC.1.3/ENV	The TSF shall initiate communication via the trusted channel for **[loading of D.App_Code, setting the Card Life Cycle State]**.

5.4 Security Requirements for the Non-IT Environment

R.ICManufacturer IC Design, manufacturing and testing

The IC manufacturer shall apply appropriate measures to ensure the confidentiality and integrity of the Smart Card Native Operating System manufacturing. This includes

- NOS source code and related data
- IC development and manufacturing proprietary information

The IC manufacturer shall apply procedures to ensure the protection of sensitive information during delivery.

There are no additional security requirements for the non-IT environment from [JCSPP].

6 TOE Summary Specification

This section provides a description of the security functions and assurance measures of the TOE that meet the TOE security requirements.

6.1 Security Functions

The following table provides a list of all security functions.

No.	TOE Security Function	Short Description	SOF
1.	SF.AccessControl	enforces the access control	high
2.	SF.Audit	Audit functionality	N/A
3.	SF.CryptoKey	Cryptographic key management	high
4.	SF.CryptoOperation	Cryptographic operation	high
5.	SF.I&A	Identification and authentication	high
6.	SF.SecureManagement	Secure management of TOE resources	high
7.	SF.PIN	PIN management	high
8.	SF.Transaction	Transaction management	N/A
9.	SF.Hardware	TSF of the underlying IC	high

Table 7: TOE Security Functions

6.1.1 SF.AccessControl

This security function ensures the access and information flow control policies of the TOE:

1 CARD CONTENT MANAGEMENT access control SFP (see sections 5.1.6.1 *FDP_ACC.1/CMGR* and 5.1.6.2 *FDP_ACF.1/CMGR*) for the import and export of user data (see sections 5.1.7.1 *FDP_ETC.1*, 5.1.7.2 *FDP_ITC.1*), loading of applet and library code (D.App_Code) and setting the card life cycle state via a trusted channel (see section 5.1.7.7 *FTP_ITC.1/CMGR*).

2 FIREWALL access control SFP (see sections 5.1.1.1 *FDP_ACC.2/FIREWALL* and 5.1.1.2 *FDP_ACF.1/FIREWALL*), and

3 JCVM information flow control SFP (see sections 5.1.1.3 *FDP_IFC.1/JCVM* and 5.1.1.4 *FDP_IFF.1/JCVM*).

It further ensures the management of the necessary security attributes:

4 Only S.PACKAGE(CM) is allowed to modify the card life cycle state (see sections 5.1.1.6 *FMT_MSA.1/CMGR*, 5.1.7.9 *FMT_SMF.1*, 5.1.1.9 *FMT_SMR.1/CMGR* and *FMT_SMR.1/JCRE)*.

5 Only the JCRE (S.JCRE) can modify the active context and the SELECTed applet Context security attributes and can change the list of registered applets' AID (see

5.1.6.3 *FMT_MSA.1/JCRE*, 5.1.4.1 *FMT_MTD.1/JCRE*, 5.1.7.9 *FMT_SMF.1*, 5.3.1.6 *FMT_SMR.1/CMGR* and *FMT_SMR.1/JCRE*.

6 Only secure values are accepted for TSF data and security attributes (see 5.3.1.4 *FMT_MSA.2/JCRE*, 5.1.4.2 *FMT_MTD.3*, 5.1.7.9 *FMT_SMF.1*, 5.1.1.9 *FMT_SMR.1/CMGR*, *FMT_SMR.1/JCRE)*. i. e.:

- The Context attribute of a *.JAVAOBJECT must correspond to that of an installed applet or be “JCRE”.
- An OB.JAVAOBJECT whose Sharing attribute is a JCRE entry point or a global array necessarily has “JCRE” as the value for its Context security attribute.
- An OB.JAVAOBJECT whose Sharing attribute value is a global array necessarily has “array of primitive Java Card System type” as a JavaCardClass security attribute’s value.
- Any OB.JAVAOBJECT whose Sharing attribute value is not “Standard” has a PERSISTENT-LifeTime attribute’s value.
- Any OB.JAVAOBJECT whose LifeTime attribute value is not PERSISTENT has an array type as JavaCardClass attribute’s value.

7 Restrictive default values are used for the security attributes, which cannot be overwritten (see 5.3.1.5 *FMT_MSA.3/CMGR* and *FMT_MSA.3/FIREWALL)*

6.1.2 SF.Audit

SF.Audit shall be able to accumulate or combine in monitoring the following auditable events and indicate a potential violation of the TSP (see 5.1.7.8):

1. Abnormal environmental conditions (frequency, voltage, temperature), in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1*, *FPT_AMT.1/SCP* and *FPT_FLS.1/JCS*.
2. Physical tampering, in fulfillment of *FAU_SAA.1*, *FPT_AMT.1/SCP*, *FPT_PHP.1* and *FPT_PHP.3/SCP*.
3. EEPROM failure audited through exceptions in the read/write operations and consistency/integrity check, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
4. Card Manager life cycle state inconsistency audited through the life cycle checks in all administrative operations and the self test mechanism on start-up, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
5. Applet life cycle inconsistency, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
6. Corruption of check-summed objects, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
7. Illegal access to the previously defined D.JAVA_OBJECT objects audited through the firewall mechanism, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.

8. Unavailability of resources audited through the object allocation mechanism, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*
9. Abortion of a transaction in an unexpected context (see (abortTransaction(), [JCAPI221] and ([JCRE222], §7.6.2), in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.

Based on the events listed above and the following events (also see 5.1.3.1):

10. Violation of the Firewall or JCVM SFPs, in fulfillment of *FAU_ARP.1/JCS* and *FPT_FLS.1/JCS*.
11. Array overflow, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
12. Other runtime errors related to applet's failure (like uncaught exceptions), in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.
13. Card tearing (unexpected removal of the Card out of the CAD) and power failure, in fulfillment of *FAU_ARP.1/JCS*, *FAU_SAA.1* and *FPT_FLS.1/JCS*.

SF.Audit shall throw an exception, lock the card session or reinitialize the Java Card System and its data upon detection of one or more of these potential security violations or respond automatically in the specified way (see 5.1.5.4) according to [ST0348].

Note: The following reactions by the TOE based on indication of a potential violation of the TSP are possible:

a) Throw an exception

b) Terminate the card (Life cycle state: TERMINATED)

c) Reinitialize the Java Card System (warm reset)

d) responding automatically according to FPT_PHP.3 ([ST0348 6.1] integrity of the EEPROM and the ROM: The EEPROM is able to correct a 1-bit error within each byte. The ROM provides a parity check. The EEPROM corrects errors automatically without user interaction, a ROM parity error forces a reset.)

e) Lock the card session (simply stops processing; escape with reset the session/Card tearing)

Based on these types of response/reaction the events listed above will have the following mapping:

Event #	Exception	Terminate card	HW Reset IC or other HW action	Lock card session
1. Abnormal environmental conditions			X	
2. Physical tampering			X	X
3.1 EEPROM failure				

audited through exceptions in the read/write operations and consistency/integrity check				X
3.2 self test mechanism on start-up				X
4. Card Manager life cycle state inconsistency audited through the life cycle checks in all administrative operations		X		
5. Applet life cycle inconsistency		X		
6. Corruption of check-summed objects				X
7. Illegal access to the previously defined D.JAVA_OBJECT objects audited through the firewall mechanism.	X			X
8. Unavailability of resources audited through the object allocation mechanism.	X			
9. Abortion of a transaction in an unexpected context	X			
10. Violation of the Firewall or JCVM SFPs	X			
11. Array overflow	X			
12. Other runtime errors related to applet's failure (like uncaught exceptions)	X			
13. Card tearing (unexpected removal of the Card out of the			X	

CAD) and power failure				

Table 8: Types of response/reaction on events

6.1.3 SF.CryptoKey

This TSF is responsible for secure cryptographic key management. Cryptographic operation is provided by the following TSF. This TSF provides the following functionality:

1. Generation of DES keys with length of 112 and 168 Bit based on random numbers according to [AIS 20] class K3 with SOF-high (see 5.1.2.1 *FCS_CKM.1*).
2. Generation of RSA keys with length from 1024 to 2368 Bit based on random numbers according to [AIS 20] class K3 with SOF-high (see 5.1.2.1 *FCS_CKM.1*).
3. Generation of AES keys with length of 128, 192, and 256 Bit based on random numbers according to [AIS 20] class K3 with SOF-high (see 5.1.2.1 *FCS_CKM.1*).
4. Distribution of DES keys with the method *setKey* of [JCAPI221] (see 5.1.2.2 *FCS_CKM.2*).
5. Distribution of RSA keys with the method *setExponent* and *setModulus* of [JCAPI221] (see 5.1.2.2 *FCS_CKM.2*).
6. Distribution of AES keys with the method *setKey* of [JCAPI221] (see 5.1.2.2 *FCS_CKM.2*).
7. Management of DES, AES, and RSA- keys with methods/commands defined in packages javacard.security and javacardx.crypto of [JCAPI221] (see 5.1.2.3 *FCS_CKM.3*).
8. Destruction of DES, AES, and RSA- keys by physically overwriting the keys by method clearKey of [JCAPI221] (see 5.1.2.4 *FCS_CKM.4*).

6.1.4 SF.CryptoOperation

This TSF is responsible for secure cryptographic operation. Cryptographic key management is provided by the previous TSF. This TSF provides the following functionality:

1. Data encryption and decryption with Triple-DES in ECB/CBC Mode and cryptographic key sizes of 112 and 168 Bit that meets [FIPS 46-3] (see 5.1.2.5 *FCS_COP.1/TripleDES*)
2. Data encryption and decryption with RSA and PKCS#1 padding [PKCS#1 v1.5]. Key sizes range from 1024 to 2368 Bit (see 5.1.2.5 *FCS_COP.1/ RSACipher*).
3. 8 byte MAC generation and verification with Triple-DES in outer CBC Mode and cryptographic key size of 112 and 168 Bit according to [ISO 9797-1] (see 5.1.2.5 *FCS_COP.1/ DESMAC*).
4. Data encryption and decryption with AES in ECB/CBC Mode and cryptographic key sizes of 128, 192, and 256 Bit that meets [FIPS 197] (see 5.1.2.5 *FCS_COP.1/ AES*).

5. RSA digital signature generation and verification with SHA-1 as hash function and cryptographic key sizes from 1024 to 2368 Bit according to [ISO 9796-2] (see 5.1.2.5 *FCS_COP.1/RSASignatureISO9796I*).
6. RSA digital signature generation and verification with SHA-1 as hash function and cryptographic key sizes from 1024 to 2368 Bit according to [PKCS#1 v1.5] (see 5.1.2.5 *FCS_COP.1/ RSASignaturePKCS#1*).
7. Secure hash computation with SHA-1 according to [FIPS 180-1] (see 5.1.2.5 *FCS_COP.1/ SHA_1*).
8. Random number generation according to [AIS 20] class K3 with SOF-high (see 5.1.7.10 *FCS_RND.1*).

6.1.5 SF.I&A

The TSF provides the following functionality with respect to card manager (administrator) authentication:

1. The TSF provide a challenge-response mechanism for card manager authentication and ensures that the session authentication data cannot be reused. After successful authentication, a trusted channel that is protected in integrity and confidentiality is established (see 5.1.7.5 *FIA_UAU.3/CMGR* and 5.1.7.6 *FIA_UAU.4/CMGR*).
2. The TSF blocks the card when 10 consecutive unsuccessful card manager authentication attempts via secure messaging using D.APP_KEY occur (see 5.1.7.3 *FIA_AFL.1/CMGR* and *FIA_AFL.1/PIN*).
3. Package execution is possible before authentication (see 5.1.6.6 *FIA_UID.1/CMGR*).

6.1.6 SF.SecureManagment

The TSF provide a secure management of TOE resources:

1. The TSF maintains a security domain for its own execution that protects it from interference and tampering by untrusted subjects. It enforces separation between the security domains of subjects in the TSC (see 5.1.5.5 *FPT_SEP.1*).
2. The TSF ensures that TSP enforcement functions are invoked and succeed before each function within the TSC is allowed to proceed (see 5.1.3.3 *FPT_RVM.1* and *FPT_RVM.1/SCP)*.
5. The TSF maintain a unique AID and version number for each package, the AID of each registered applet, and whether a registered applet is currently selected for execution ([JCVM221], §6.5) (see 5.1.4.3 *FIA_ATD.1/AID*, 5.1.4.4 *FIA_UID.2/AID* and 5.1.4.5 *FIA_USB.1*).
6. The TSF run a suite of self-tests during initial start-up (at each power on) to demonstrate the correct operation of the TSF, to verify the integrity of TSF data, and to verify the integrity of stored TSF executable code. This includes checking the EEPROM integrity. If an error is detected, the TOE enters into a secure state (lock card session) (see 5.1.5.2 *FPT_FLS.1/SCP*, 5.1.5.4 *FPT_PHP.1* and 5.1.6.3 *FPT_TST.1*).

7. The TSF ensures that packages are unable to observe operations on secret keys and PIN codes by other subjects (see 5.1.3.5 *FPR_UNO.1*).
8. The TSF monitors user data D.APP_CODE, D.APP_I_DATA, D.PIN, D.APP_KEYs for integrity errors. If an error occurs, the TSF maintain a secure state (lock card session) (see 5.1.3.2 *FDP_SDI.2, 5.1.5.6 FPT_RCV.3/SCP*).
9. The TSF makes any previous information content of a resource unavailable upon (see 5.1.2.6 *FDP_RIP.1/OBJECTS*, *FDP_RIP.1/APDU*, *FDP_RIP.1/bArray*, *FDP_RIP.1/TRANSIENT*, *FDP_RIP.1/ABORT* and *FDP_RIP.1/KEYS*):
    - allocation of class instances, arrays, and the APDU buffer,
    - de-allocation of bArray object, any transient object, any reference to an object instance created during an aborted transaction, and cryptographic buffer (D.CRYPTO).
10. The TSF ensures that during command execution there are no usable variations in power consumption (measurable at e. g. electrical contacts) or timing (measurable at e. g. electrical contacts) that might disclose cryptographic keys or PINs.[34] All functions of SF.CryptoOperation except with SHA-1 are resistant to side-channel attacks (e.g. timing attack, SPA, DPA, DFA, EMA, DEMA) (see 5.1.7.11 *FPT_EMSEC.1*).
11. CAP files, the bytecode and its data arguments are consistently interpreted using the following rules (see 5.1.7.15 *FPT_TDC.1*):
    a. The [JCVM221] specification;
    b. Reference export files;
    c. The ISO 7816-6 rules;
    d. The EMV specification.

6.1.7 SF.PIN

The TSF provides the following functionality with respect to user authentication with the global PIN (D.PIN):

1. The TSF provide user authentication with a Global-PIN that is at least 6 digits long (see 5.1.7.4 *FIA_UAU.1*).
2. The maximum possible number of consecutive unsuccessful PIN-authentication attempts is user configurable number from 1 to 127. (see 5.1.7.3 *FIA_AFL.1/CMGR*, *FIA_AFL.1/PIN*)

   Note: For SOF-high, the maximum number must be limited to 3. That fact must be mentioned in the guidance.
3. When this number has been met or surpassed, the PIN-authentication is blocked (see 5.1.7.3 *FIA_AFL.1/CMGR* and *FIA_AFL.1/PIN*).

[34] Note: All measures described in guidance of the underlying hardware platform concerning power consumption and timing will be taken into account for the TOE development.

4. Only the following commands are allowed, before successful authentication (see 5.1.7.4 *FIA_UAU.1*):
   - *Get Data* with objects: ISD DATA [ISSUER IDENTIFICATION NUMBER], ISD DATA [CARD IMAGE NUMBER], PLATFORM DATA [CARD RECOGNITION DATA], ISD DATA [KEY INFORMATION TEMPLATE], ISD DATA [SCP INFORMATION], PLATFORM DATA [MANUFACTURING]
   - *Select Applet*
   - *Initialize Update* with object: APDU BUFFER
   - *External Authenticate* with object: APDU BUFFER

6.1.8 SF.Transaction

The TSF permits the rollback of operations OP.JAVA, OP.CREATE on objects OB.JAVAOBJECTs. These operations can be rolled back within the calls: select(), deselect(), process() or install(), notwithstanding the restrictions given in [JCRE221], §7.7, within the bounds of the Commit Capacity ([JCRE221], §7.8), and those described in [JCAPI221]. (see 5.1.2.7 *FDP_ROL.1/FIREWALL, 2, 5.1.5.6 FPT_RCV.4/SCP*).

6.1.9 SF.Hardware

The certified hardware (part of the TOE) features the following TSF. The exact formulation can be found in [ST0348]:

1. Random Number Generator (F.RNG) (see 5.1.7.10 *FCS_RND.1*).
2. Triple-DES Co-processor (F.HW_DES) (see 5.1.2.5 *FCS_COP.1/TripleDES*).
3. AES Co-processor (F.HW_AES) (see 5.1.2.5 *FCS_COP.1/AES*).
4. Control of Operating Conditions (F.OPC) (see 5.1.3.4 *FPT_FLS.1/SCP*, 5.1.5.5 *FPT_SEP.1/SCP*, 5.1.5.3 *FRU_FLT.2/SCP*).
5. Protection against Physical Manipulation (F.PHY) (see 5.1.2.5 *FCS_COP.1/TripleDES* and *FCS_COP.1/AES*, 5.1.7.10 *FCS_RND.1*, 5.1.3.4 *FPT_FLS.1/SCP*, 5.1.7.14 *FPT_PHP.1*, 5.1.5.4 *FPT_PHP.3/SCP*, 5.1.5.5 *FPT_SEP.1/SCP* and 5.1.5.3 *FRU_FLT.2/SCP*).
6. Logical Protection (F.LOG) (see 5.1.7.11 *FPT_EMSEC.1*)
7. Protection of Mode Control (F.COMP) (see 5.1.7.12 *FMT_LIM.1*, 5.1.7.13 *FMT_LIM.2* and 5.1.5.5 *FPT_SEP.1/SCP*).
8. Memory Access Control (F.MEM_ACC), not necessary to fulfill any SFR.
9. Special Function Register Access Control (F.SFR_ACC), , not necessary to fulfill any SFR.

6.2 Strength of Function Claims

The following functions have a **SOF-high** claim:

- SF.AccessControl (aspect 1)
- SF.CryptoKey (aspects 1, 2, 3)
  The quality of random numbers used for key generation is expressed as a metric according to AIS 20 class K3, SOF-high.
- SF.CryptoOperation (aspects 7, 8)
  aspect 8 is expressed as a metric according to AIS 20 class K3, SOF-high
- SF.I&A (aspects 1, 2)
- SF.SecureManagement (aspect 6)
- SF.PIN (aspects 1, 2, 3)
- SF.Hardware (aspects 1, 6)

All other security functions (SF.Audit, SF.Transaction) are not based on probability or permutational mechanisms. Furthermore, cryptographic aspects of SF.CryptoKey and SF.CryptoOperation are outside the scope of CC evaluations.

6.3 Assurance Measures

The TOE is to fulfill the assurance requirements of assessment class ASE and of evaluation level EAL4 augmented by ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3. The present document "Security Target" serves to fulfill the requirements according to ASE. Besides the TOE (according to ATE_IND.2), the manufacturer will provide the following additional documents within the frame of the evaluation, to evidently prove the fulfilling of the requirements according to EAL4 augmented by ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3:

- Configuration management documentation (according to ACM_AUT.1 and ACM_CAP.4)
- Delivery and operational documentation (according to ADO_DEL.2 and ADO_IGS.1)
- Functional specification documentation (according to ADV_FSP.2)
- High-level design documentation (according to ADV_HLD.2)
- Implementation representation documentation (according to ADV_IMP.2)
- Low-level design documentation (according to ADV_LLD.1)
- Representation correspondence documentation (according to ADV_RCR.1)
- Security policy modeling documentation (according to ADV_SPM.1)
- Guidance documents documentation (according to AGD_ADM.1 and AGD_USR.1)

- Life cycle support documentation (according to ALC_DVS.2, ALC_LCD.1, and ALC_TAT.1)
- Tests documentation (according to ATE_COV.2, ATE_DPT.1, and ATE_FUN.1)
- Vulnerability assessment documentation (according to AVA_MSU.3, AVA_SOF.1, and AVA_VLA.4)

The assignment of the assurance measures to the assurance requirements (see section 5.2) is straight forward, as for all assurance components (with exception of the independent testing of the evaluator ATE_IND.2) documentation is provided.

7 PP Claims

7.1 PP Reference

This security target (ST) is based on the following protection profile:

- Java Card System – Minimal Configuration Protection Profile, Version: 1.0b, August 2003 [JCSPP]

This ST makes claims for formal conformance to this PP, as the ST fulfils all requirements of [JCSPP]. This ST even chooses a hierarchically higher augmentation of EAL4, in comparison to [JCSPP], by selecting ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3.

Further assumptions, threats, one organizational security policy, security objectives, and IT security requirements not contained in [JCSPP] were defined in this ST and marked that they were not taken from [JCSPP].

7.2 PP Additions and Refinements

Additions and refinements of chapter 3 and 4 of this ST are mentioned in:

- Table 1: Assumptions
- Table 2: Threats
- Table 3: Organizational Security Policies
- Table 4: Security Objectives for the TOE, and
- Table 5: Security Objectives for the environment

The following SFRs have been added compared to SFR for the TOE defined in the *Java Card System – Minimal Configuration Protection Profile* [JCSPP]:

- from **[PP0002]**: FCS_RND.1
- from **[PP0017]**: FPT_EMSEC.1, FMT_LIM.1, FMT_LIM.2
- from **[JCSPP] SCP group**: FPT_AMT.1/SCP, FPT_FLS.1/SCP, FRU_FLT.2/SCP, FPT_PHP.3/SCP, FPT_SEP.1/SCP, FPT_RCV.3/SCP, FPT_RCV.4/SCP and FPT_RVM.1/SCP
- from **[JCSPP] CMGR group**: FDP_ACC.1/CMGR, FDP_ACF.1/CMGR, FMT_MSA.1/CMGR, FMT_MSA.3/CMGR, FMT_SMR.1/CMGR, and FIA_UID.1/CMGR
- from **CC part 2**: FAU_SAA.1, FMT_SMF.1, FDP_ETC.1, FDP_ITC.1, FIA_AFL.1/PIN, FIA_UAU.1, FIA_UAU.3, FIA_UAU.4, FIA_AFL.1/CMGR, FTP_ITC.1/ENV, FPT_PHP.1 and FPT_TDC.1

There were refinements and editorial changes of the SFR of [JCSPP] as follows:

SFR	Chapter in ST	Changes
FDP_ACF.1.1/FIREWALL	5.1.1.2	Refinement due to FI103 (already incorporated in CC used for evaluation)
FDP_ACF.1.4/FIREWALL	5.1.1.2	Editorial refinement
FDP_IFF.1.1/JCVM	5.1.1.4	Refinement for types of subjects due to FI104 (already incorporated in CC used for evaluation)
FDP_IFF.1.2/JCVM	5.1.1.4	Editorial refinement
FMT_MSA.1.1/JCRE	5.1.1.6	Editorial refinement
FMT_MSA.3.2/FIREWALL	5.1.1.8	Editorial refinement
FPT_TST.1.1	5.1.3.6	Editorial refinement due to FI056.
FPT_TST.1.2	5.1.3.6	Editorial refinement due to FI056
FIA_USB.1.1	5.1.4.5	Editorial refinement; According to final interpretation #137 the SFR FIA_USB.1 is rewritten
FRU_FLT.2	5.1.5.3	Usage of hierarchical component
FPT_PHP.3.1	5.1.5.4	Editorial refinement
FPT_SEP.1.2	5.1.5.5	Editorial refinement
FPT_RCV.3.1	5.1.5.6	Editorial refinement due to FI056.
FMT_SMF.1	5.1.7.9	Added due to FI065 (already incorporated in CC used for evaluation)
FDP_IFF.2.2/BCV	5.3.1.2	Editorial refinement
FMT_MSA.3.2/BCV	5.3.1.5	Editorial refinement.
FMT_SMF.1/BCV	5.3.1.7	Added due to FI065 (already incorporated in CC used for evaluation)

Table 9: Refinements on SFR taken from [JCSPP]

The assurance level for this ST is EAL4 augmented by ADV_IMP.2, **ALC_DVS.2,** AVA_VLA.4 **and AVA_MSU.3** compared to the assurance requirements of *EAL4*

augmented by *ADV_IMP.2* and *AVA_VLA.3 (SOF-medium)* of [JCSPP]. Therefore, this ST fully satisfies the assurance requirements specified in [JCSPP].

The following table lists all application notes from [JCSPP] relevant for this ST and how the application notes have been applied. **Bold type** maps the parts which are corresponding and different.

Application notes from [JCSPP]	Refined / followed by ST
Not in [JCSPP]	[ST 3.3.1.3] This is to assume that the keys used in terminals or systems are correctly protected for confidentiality and integrity in their own environment, as the disclosure of such information which is shared with the TOE but is not under the TOE control, may compromise the security of the TOE.
Not in [JCSPP]	[ST 4.2.1.6] Objectives for the TOE environment are usually not satisfied by the TOE Security Functional Requirements. The TOE development and manufacturing environment (phases 1 to 3) is in the scope of this ST. These phases are under the TOE developer scope of control. Therefore, the objectives for the environment related to phase 1 to 3 are covered by Assurance measures, which are materialized by documents, process and procedures evaluated through the TOE evaluation process. The `product usage phases` (phase 4 to 7) are not in the scope of the evaluation. During these phases, the TOE is no more under the developer control. In this environment, the TOE protects itself with its own Security functions. But some additional usage recommendation must also be followed in order to ensure that the TOE is correctly and securely handled, and that shall be not damaged or compromised. This ST assumes (A.DLV_DATA, A.TEST_OPERATE, A.USE_DIAG, A.USE_KEYS) that users handle securely the TOE and related Objectives for the environment are defined (OE.DLV_DATA, OE.TEST_OPERATE, OE.USE_DIAG, OE.USE_KEYS)
[JCSPP 4.1.1] Application note: To be made unavailable means to be physically erased with a default value. Except for local variables that do not correspond to method parameters, the default values to be used are specified in	[ST 4.1.2.2] Application note:To be made unavailable means to be physically erased with a default value. Except for local variables that do not correspond to method parameters, the default values to be used are specified in [JCVM221].

Application notes from [JCSPP]	Refined / followed by ST
[JCVM21].	
[JCSPP 4.1.1] Application note: PIN objects may play key roles in the security architecture of client applications. The way they are stored and managed in the memory of the smart card must be carefully considered, and this applies to the whole object rather than the sole value of the PIN. For instance, the try counter's value is as sensitive as that of the PIN.	[ST 4.1.2.3] Application note:PIN objects may play key roles in the security architecture of client applications. The way they are stored and managed in the memory of the smart card must be carefully considered, and this applies to the whole object rather than the sole value of the PIN. For instance, the try counter's value is as sensitive as that of the PIN.
[JCSPP 4.1.1] Application note: O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION and O.CIPHER are actually provided to applet s in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to applet s; those may be built on top of the Java Card API or independently. **Depending on whether they contain native code or not, these proprietary libraries will need to be evaluated together with the TOE or not (see #.NATIVE, p.40). In any case, they are not included in the Java Card System for the purpose of the present document.**	[ST 4.1.2.3] Application note:O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION and O.CIPHER are actually provided to applets in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to applets; those may be built on top of the Java Card API or independently. **Note: For this Java Card such libraries do not exist. All necessary functionality is implemented by the TOE.**
[JCSPP 4.1.3] Application note: Usurpation of identity resulting from a malicious installation of an applet on the card may also be the result of perturbing the communication channel linking the CAD and the card. Even if the CAD is placed in a secure environment, the attacker may try to capture, duplicate, permute or modify the packages sent to the card. He may also try to send one of its own applications as if it came from the card issuer . Thus, this objective is intended to ensure the integrity and authenticity of loaded CAP files.	**Not included.**
[JCSPP 5.1.1.1] Application note: The deletion of applet s may render some *O.JAVAOBJECT* inaccessible, and the JCRE may be in charge of this aspect. This can be done, for instance, by ensuring that references to objects belonging to a deleted application are considered as a null reference. Such a mechanism is implementation-dependent.	[ST 5.1.1.2] Application note:The deletion of applets may render some OB.JAVAOBJECT inaccessible, and the JCRE may be in charge of this aspect. This can be done, for instance, by ensuring that references to objects belonging to a deleted application are considered as a null reference. Such a mechanism is implementation-dependent.

Application notes from [JCSPP]	Refined / followed by ST
[JCSPP 5.1.1.1] Application note: References of temporary JCRE entry points , which cannot be stored in class variables, instance variables or array components, are transferred from the internal memory of the JCRE (TSF data) to some stack through specific APIs (JCRE owned exceptions) or JCRE invoked methods (such as the process(APDU apdu)); these are causes of *OP.PUT(S1*,*S2*,*I)* operations as well.	[ST 5.1.1.3] Application note:References of temporary JCRE entry points, which cannot be stored in class variables, instance variables or array components, are transferred from the internal memory of the JCRE (TSF data) to some stack through specific APIs (JCRE owned exceptions) or JCRE invoked methods (such as the process(APDU apdu)); these are causes of OP.PUT(S1,S2,I) operations as well.
[JCSPP 5.1.1.1] Application note: the storage of temporary JCRE-owned objects' references is runtime-enforced ([**JCRE21**], §6.2.8.1-3).	[ST 5.1.1.4] Application note:the storage of temporary JCRE-owned objects' references is runtime-enforced ([**JCRE221**], §6.2.8.1-3).
[JCSPP 5.1.1.1] Application note: The semantics of the Java programming language requires for any object field and array position to be initialized with default values when the resource is allocated [JVM],§2.5.1.	[ST 5.1.1.5] Application note:The semantics of the Java programming language requires for any object field and array position to be initialized with default values when the resource is allocated [JVM],§2.5.1.
[JCSPP 5.1.1.1] Application note: The modification of the active context as well as that of the selected applet should be performed in accordance with the rules given in [**JCRE21**], §4 and [**JCVM21**], §3.4.	[ST 5.1.1.6] Application note:The modification of the active context as well as that of the selected applet should be performed in accordance with the rules given in [**JCRE221**], §4 and [**JCVM221**], §3.4.
[JCSPP 5.1.1.1] Application note: For instance, secure values conform to the following rules: – The Context attribute of a **.JAVAOBJECT*10 must correspond to that of an installed applet or be "JCRE". – An ***O****.JAVAOBJECT* whose Sharing attribute is a JCRE entry point or a global array necessarily has "JCRE" as the value for its Context security attribute. – An ***O****.JAVAOBJECT* whose Sharing attribute value is a global array necessarily has "array of primitive Java Card System type" as a JavaCardClass security attribute's value. – Any ***O****.JAVAOBJECT* whose Sharing attribute value is not "Standard" has a PERSISTENT-LifeTime attribute's value. – Any ***O****.JAVAOBJECT* whose LifeTime attribute value is not PERSISTENT has an array type as JavaCardClass attribute's value.	[ST 5.1.1.7] Application note:For instance, secure values conform to the following rules: – The Context attribute of a *.JAVAOBJECT must correspond to that of an installed applet or be "JCRE". – An **OB**.JAVAOBJECT whose Sharing attribute is a JCRE entry point or a global array necessarily has "JCRE" as the value for its Context security attribute. – An **OB**.JAVAOBJECT whose Sharing attribute value is a global array necessarily has "array of primitive Java Card System type" as a JavaCardClass security attribute's value. – Any **OB**.JAVAOBJECT whose Sharing attribute value is not "Standard" has a PERSISTENT-LifeTime attribute's value. – Any **OB**.JAVAOBJECT whose LifeTime attribute value is not PERSISTENT has an array type as JavaCardClass attribute's value.

Application notes from [JCSPP]	Refined / followed by ST
[JCSPP 5.1.1.1] Application note: The above rules are given as examples only. For instance, the last two rules are motivated by the fact that the Java Card API defines only transient arrays factory methods. Future versions may allow the creation of transient objects belonging to arbitrary classes; such evolution will naturally change the range of "secure values" for this component.	[ST 5.1.1.7] Application note: The above rules are given as examples only. For instance, the last two rules are motivated by the fact that the Java Card API defines only transient arrays factory methods. Future versions may allow the creation of transient objects belonging to arbitrary classes; such evolution will naturally change the range of "secure values" for this component.
[JCSPP 5.1.1.1] Application note: Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (*FMT_MSA.1/JCRE*). At the creation of an object (*OP.CREATE*), the newly created object, assuming that the operation is permitted by the SFP, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([**JCRE21**], §6.1.2). There is one default value for the *SELECTed applet Context* that is the *default applet identifier's Context*, and one default value for the *active context*, that is "*JCRE*".	[ST 5.1.1.8] Application note: Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (FMT_MSA.1/JCRE). At the creation of an object (OP.CREATE), the newly created object, assuming that the operation is permitted by the SFP, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([**JCRE221**], §6.1.2). There is one default value for the SELECTed applet Context that is the default applet identifier's Context, and one default value for the active context, that is "JCRE".
[JCSPP 5.1.1.1] Application note: There is no security attribute attached to subjects or information for this information flow policy. However, this is the JCRE who controls the currently active context. Moreover, the knowledge of which reference corresponds to a temporary entry point object or a global array and which does not is solely available to the JCRE (and the virtual machine).	[ST 5.1.1.8] Application note: There is no security attribute attached to subjects or information for this information flow policy. However, this is the JCRE who controls the currently active context. Moreover, the knowledge of which reference corresponds to a temporary entry point object or a global array and which does not is solely available to the JCRE (and the virtual machine).
[JCSPP 5.1.1.1] Application note: The intent is that none of the identified roles has privileges with regard to the default values of the security attributes. Notice that creation of objects is an operation controlled by the *FIREWALL SFP*; the latitude on the parameters of this operation is described there. The operation shall fail anyway if the created object would have had security attributes whose value violates	[ST 5.1.1.8] Application note: The intent is that none of the identified roles has privileges with regard to the default values of the security attributes. Notice that creation of objects is an operation controlled by the FIREWALL SFP; the latitude on the parameters of this operation is described there. The operation shall fail anyway if the created object would have had security attributes whose value violates FMT_MSA.2.1/JCRE.

Application notes from [JCSPP]	Refined / followed by ST
FMT_MSA.2.1/JCRE.	
[JCSPP 5.1.1.1] Application note: By security domain it is intended “execution context” which should not be confused with other meanings of “security domains”.	[ST 5.1.1.9] Application note:By security domain it is intended “execution context” which should not be confused with other meanings of “security domains”.
[JCSPP 5.1.1.2] Application note: The keys can be generated and diversified in accordance with [**JCAPI21**] specification in classes KeyBuilder and KeyPair (at least Session key generation).	[ST 5.1.2.1] Application note:The keys can be generated and diversified in accordance with [**JCAPI221**] specification in classes KeyBuilder and KeyPair (at least Session key generation).
[JCSPP 5.1.1.2] Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22] for 2.2, [JCAPI21] for 2.1).	**Not included [ST 5.1.2.1].**
[JCSPP 5.1.1.2] Application note: Command SetKEY that meets [JCAPI21] standard.	**Not included [ST 5.1.2.2].**
[JCSPP 5.1.1.2] Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22] for 2.2, [JCAPI21] for 2.1).	**Not included [ST 5.1.2.2].**
[JCSPP 5.1.1.2] Application note: The keys can be accessed in accordance with [**JCAPI21**] in class Key.	[ST 5.2.1.3] Application note:The keys can be accessed in accordance with [**JCAPI221**] in class Key.
[JCSPP 5.1.1.2] Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22] for 2.2, [JCAPI21] for 2.1).	**Not included [ST 5.1.2.3].**
[JCSPP 5.1.1.2] Application note: The keys are reset in accordance with **[JCAPI21]** in class Key with the method clearKey(). Any access to a cleared key attempting to use it for ciphering or signing shall throw an exception.	[ST 5.1.2.4] Application note:The keys are reset in accordance with **[JCAPI221]** in class Key with the method clearKey(). Any access to a cleared key attempting to use it for ciphering or signing shall throw an exception.
[JCSPP 5.1.1.2] Application note: The TOE shall provide a subset of cryptographic operations defined in [JCAPI21] in accordance to [JCAPI21] specification (see javacardx.crypto.Cipher and javacardx.security packages).	**Note is missing in [ST 5.1.2.5], however, defined are:** **FCS_COP.1/TripleDES** **FCS_COP.1/AES** **FCS_COP.1/RSACipher** **FCS_COP.1/DESMAC** **FCS_COP.1/RSASignatureISO9796**

Application notes from [JCSPP]	Refined / followed by ST
	FCS_COP.1/RSASignaturePKCS#1 **FCS_COP.1/SHA-1**
[JCSPP 5.1.1.2] Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22] for 2.2, [JCAPI21] for 2.1).	**Not included [ST 5.1.2.4].**
[JCSPP 5.1.1.2] Application note: The allocation of a resource to the APDU buffer is typically performed as the result of a call to the process() method of an applet.	[ST 5.2.1.6] Application note:The allocation of a resource to the APDU buffer is typically performed as the result of a call to the process() method of an applet.
[JCSPP 5.1.1.2] Application note: A resource is allocated to the bArray object when a call to *an* applet's install() method is performed. There is no conflict with *FDP_ROL.1* here because of the bounds on the rollback mechanism (*FDP_ROL.1.2/FIREWALL*): the scope of the rollback does not extend outside the execution of the install() method, and the de-allocation occurs precisely right after the return of it.	[ST 5.1.2.6] Application note:A resource is allocated to the bArray object when a call to an applet's install() method is performed. There is no conflict with FDP_ROL.1 here because of the bounds on the rollback mechanism (FDP_ROL.1.2/FIREWALL): the scope of the rollback does not extend outside the execution of the install() method, and the de-allocation occurs precisely right after the return of it.
[JCSPP 5.1.1.2] Application note: The events that provoke the de-allocation of a transient object are described in **[JCRE21]***, §5.1.*	[ST 5.1.2.6] Application note:The events that provoke the de-allocation of a transient object are described in **[JCRE221]**, §5.1.
[JCSPP 5.1.1.2] Application note: The events that provoke the de-allocation of the previously mentioned references are described in [**JCRE21**], §7.6.3.	[ST 5.1.2.6] Application note:The events that provoke the de-allocation of the previously mentioned references are described in [**JCRE221**], §7.6.3.
[JCSPP 5.1.1.2] Application note: The javacard.security & javacardx.crypto package s do provide secure interfaces to the *cryptographic buffer* in a transparent way. See javacard.security.KeyBuilder and Key interface of [**JCAPI21**].	[ST 5.1.2.6] Application note:The javacard.security & javacardx.crypto packages do provide secure interfaces to the cryptographic buffer in a transparent way. See javacard.security.KeyBuilder and Key interface of [**JCAPI221**].
[JCSPP 5.1.1.2] Application note: Java Card System 2.1.1 defines no explicit (or implicit) deallocation of objects, but those caused by the failure of installation or the abortion of a transaction. The only related function for keys is the clearKey() method, which does not mandate erasure of the contents of the key (see *FCS_CKM.4*) nor the behavior of the transaction with respect to this "clearing". ST authors may consider additional	[ST 5.1.2.6] Application note:Java Card System 2.1.1 defines no explicit (or implicit) de-allocation of objects, but those caused by the failure of installation or the abortion of a transaction. The only related function for keys is the clearKey() method, which does not mandate erasure of the contents of the key (see FCS_CKM.4) nor the behavior of the transaction with respect to this "clearing". ST authors may consider additional security requirements on this

Application notes from [JCSPP]	Refined / followed by ST
security requirements on this topic.	topic.
[JCSPP 5.1.1.2] Application note: Transactions are a service offered by the APIs to applet s. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [**JCAPI21**] (see for instance, PIN-blocking, PINchecking, update of Transient object s).	[ST 5.1.2.7] Application note:Transactions are a service offered by the APIs to applets. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [**JCAPI221**] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).
[JCSPP 5.1.1.2] Application note: The loading and linking of applet package s (the installation or registration is covered by *FDP_ROL.1.1/FIREWALL*) is subject to some kind of rollback mechanism (see *FPT_RCV.3.1/Installer*), described in [**JCRE21**], §10.1.4, but is implementation-dependent.	[ST 5.1.2.7] Application note:The loading and linking of applet packages (the installation or registration is covered by FDP_ROL.1.1/FIREWALL) is subject to some kind of rollback mechanism, described in [**JCRE221**], §10.1.4, but is implementation-dependent.
[JCSPP 5.1.1.3] Application note: The thrown exceptions and their related events are described in **[JCRE21], [JCAPI21], and [JCVM21]**.	[ST 5.1.3.1] Application note:The thrown exceptions and their related events are described in **[JCRE221], [JCAPI221], and [JCVM221]**.
[JCSPP 5.1.1.3] Application note: The bytecode verification defines a large set of rules used to detect a “potential security violation”. The actual monitoring of these “events” within the TOE only makes sense when the bytecode verification is performed on-card.	[ST 5.1.3.1] Application note:The bytecode verification defines a large set of rules used to detect a “potential security violation”. The actual monitoring of these “events” within the TOE only makes sense when the bytecode verification is performed on-card.
[JCSPP 5.1.1.3] Application note: Depending on the context of use and the required security level, there are cases where the card manager and the TOE must work in cooperation to detect and appropriately react in case of potential security violation. This behavior must be described in this component. It shall detail the nature of the feedback information provided to the card manager (like the identity of the offending application) and the conditions under which the feedback will occur (any occurrence of the java.lang.SecurityException exception).	[ST 5.1.3.1] Application note:Depending on the context of use and the required security level, there are cases where the card manager and the TOE must work in cooperation to detect and appropriately react in case of potential security violation. This behavior must be described in this component. It shall detail the nature of the feedback information provided to the card manager (like the identity of the offending application) and the conditions under which the feedback will occur (any occurrence of the java.lang.SecurityException exception).
[JCSPP 5.1.1.3] Application note: The “locking of the card session” may not appear in the policy of the card manager.	[ST 5.1.3.1] Application note:The “locking of the card session” may not appear in the policy of the card manager. Such measure

Application notes from [JCSPP]	Refined / followed by ST
Such measure should only be taken in case of severe violation detection; the same holds for the re-initialization of the Java Card System. Moreover, the locking should occur when “clean” re-initialization seems to be impossible.	should only be taken in case of severe violation detection; the same holds for the re-initialization of the Java Card System. Moreover, the locking should occur when “clean” re-initialization seems to be impossible.
[JCSPP 5.1.1.3] Application note: Although no such requirement is mandatory in the specification, at least an exception shall be raised upon integrity errors detection on cryptographic keys, PIN values and their associated security attributes. Even if all the objects cannot be monitored, cryptographic keys and PIN objects shall be considered with particular attention by ST authors as they play a key role in the overall security.	**Not included [ST 5.1.3.2].**
[JCSPP 5.1.1.3] Application note: It is also recommended to monitor integrity errors in the code of the native applications and Java Card technology-based applications (“Java Card applications”).	**Not included [ST 5.1.3.2].**
[JCSPP 5.1.1.3] Application note: Execution of native code is not within the TSC. Nevertheless, access to native methods from the Java Card System is subject to TSF control, as there is no difference in the interface or the invocation mechanism between native and interpreted methods.	[ST 5.1.3.3] Application note: Execution of native code is not within the TSC. Nevertheless, access to native methods from the Java Card System is subject to TSF control, as there is no difference in the interface or the invocation mechanism between native and interpreted methods.
[JCSPP 5.1.1.3] Application note: Concerning the interpretation of data between the TOE and the underlying Java Card platform, it is assumed that the TOE is developed consistently with the SCP functions, namely concerning memory management, I/O functions, cryptographic functions, and so on.	**Not included [ST 5.1.7.15].**
[JCSPP 5.1.1.3] Application note: The JCRE Context is the Current context when the VM begins running after a card reset ([**JCRE21**], §6.2.3). Behavior of the TOE on power loss and reset is described in [**JCRE21**], §3.5, and §7.1.	[ST 5.1.3.4] Application note: The JCRE Context is the Current Context when the VM begins running after a card reset ([**JCRE221**], §6.2.3). Behavior of the TOE on power loss and reset is described in [**JCRE221**], §3.5, and §7.1.
[JCSPP 5.1.1.3] Application note: Although it is not required in [**JCRE21**] specifications, the nonobservability of operations on sensitive information such as keys appears as impossible to	[ST 5.1.3.5] Application note: Although it is not required in [**JCRE221**] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart

Application notes from [JCSPP]	Refined / followed by ST
circumvent in the smart card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.	card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.
[JCSPP 5.1.1.3] Application note: TSF-testing is not mandatory in [**JCRE21**], but appears in most of security requirements documents for masked applications. Testing could also occur randomly.	[ST 5.1.3.6] Application note:TSF-testing is not mandatory in [**JCRE221**], but appears in most of security requirements documents for masked applications. Testing could also occur randomly.
[JCSPP 5.1.1.4] Application note: The installer and the JCRE manage some other TSF data such as the applet life cycle or CAP file s, but this management is implementation specific. Objects in the Java programming language may also try to query AIDs of installed applet s through the lookupAID(…) API method.	[ST 5.1.4.1] Application note:The installer and the JCRE manage some other TSF data such as the applet life cycle or CAP files, but this management is implementation specific. Objects in the Java programming language may also try to query AIDs of installed applets through the lookupAID(…) API method.
[JCSPP 5.1.1.4] Application note: The installer, applet deletion manager or even the card manager may be granted the right to modify the list of registered applets' AIDs in specific implementations (possibly needed for installation and deletion; see #.DELETION and #.INSTALL).	[ST 5.1.4.1] Application note:The installer, applet deletion manager or even the card manager may be granted the right to modify the list of registered applets' AIDs in specific implementations (possibly needed for installation and deletion; see #.DELETION and #.INSTALL).
[JCSPP 5.1.1.4] Application note: By users here it must be understood the ones associated to the packages (or applets) which act as subjects of policies. In the Java Card System, every action is always performed by an identified user interpreted here as the currently selected applet or the package that is the subject 's owner. Means of identification are provided during the loading procedure of the package and the registration of applet instances.	[ST 5.1.4.4] Application note:By users here it must be understood the ones associated to the packages (or applets) which act as subjects of policies. In the Java Card System, every action is always performed by an identified user interpreted here as the currently selected applet or the package that is the subject's owner. Means of identification are provided during the loading procedure of the package and the registration of applet instances.
[JCSPP 5.1.1.4] Application note: The role JCRE defined in *FMT_SMR.1/JCRE* is attached to an IT security function rather than to a "user" of the CC terminology. The JCRE does not "identify" itself with respect to the TOE, but it is a part of it.	[ST 5.1.4.4] The role *JCRE* defined in *FMT_SMR.1/JCRE* is attached to an IT security function rather than to a "user" of the CC terminology. The *JCRE* does not "identify" itself with respect to the TOE, but it is a part of it.
[JCSPP 5.1.1.4] Application note: For *S.PACKAGE*s, the Context security attribute plays the role of the appropriate	[ST 5.1.4.5] For *S.PACKAGE*S, the Context security attribute plays the role of the appropriate user security attribute; see

Application notes from [JCSPP]	Refined / followed by ST
user security attribute; see *FMT_MSA.1.1/JCRE* **below**.	*FMT_MSA.1.1/JCRE* **above**.
[JCSPP 5.1.2] Application note: The most common importation of user data is package loading and applet installation on the behalf of the installer . Security attributes consist of the shareable flag of the class component, AID and version numbers of the package, maximal operand stack size and number of local variables for each method, and export and import components (visibility).	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: The format of the CAP file is precisely defined in Sun's specification ([JCVM21]); it contains the user data (like applet's code and data) and the security attribute altogether. Therefore there is no association to be carried out elsewhere.	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: Each package contains a package Version attribute, which is a pair of major and minor version numbers ([JCVM21], §4.5). With the AID, it describes the package defined in the CAP file. When an export file is used during preparation of a CAP file, the versions numbers and AIDs indicated in the export file are recorded in the CAP files ([JCVM21], §4.5.2): the dependent packages Versions and AIDs attributes allow the retrieval of these identifications.. Implementationdependent checks may occur on a case-by-case basis to indicate that package files are binary compatibles. However, package files do have "package Version Numbers" ([JCVM21]) used to indicate binary compatibility or incompatibility between successive implementations of a package, which obviously directly concern this requirement.	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: The installation (the invocation of an applet 's install method by the installer) is implementation dependent ([JCRE21]§10.2).	**[Applet installation not covered by the ST]**

Application notes from [JCSPP]	Refined / followed by ST
[JCSPP 5.1.2] Application note: Other rules governing the installation of an applet , that is, its registration to make it SELECTable by giving it a unique AID , are also implementation dependent (see, for example, [JCRE21], §10).	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: The TOE may provide additional feedback information to the card manager in case of potential security violations (see *FAU_ARP.1*).	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: This element is not within the scope of the Java Card specification, which only mandates the behavior of the Java Card System in good working order. Further details on the “maintenance mode” shall be provided in specific implementations. The following is an excerpt from [CC1]: In this maintenance mode normal operation might be impossible or severely restricted, as otherwise nsecure situations might occur. Typically, only authorized users should be allowed access to this mode but the real details of who can access this mode is a function of class FMT Security management. If FMT does not put any controls on who can access this mode, then it may be acceptable to allow any user to restore the system if the TOE enters such a state. However, in practice, this is probably not desirable as the user restoring the system has an opportunity to configure the TOE in such a way as to violate the TSP.	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: Should the installer fail during loading/installation of a package/applet , it has to revert to a “consistent and secure state”. The JCRE has some clean up duties as well; see [JCRE21], §10.1.4 for possible scenarios. Precise behavior is left to implementers.	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: In the case where the configuration includes the applet deletion manager (and the associated group, *ADELG*), this component shall include among the listed failures that of the deletion of a package/applet . See ([JCRE22], 11.3.4) for possible scenarios. Precise behavior is left to implementers.	**[Applet installation not covered by the ST]**
[JCSPP 5.1.2] Application note: The	**[Applet installation not covered by the**

Application notes from [JCSPP]	Refined / followed by ST
quantification is implementation dependent, but some facts can be recalled here. First, the SCP ensures the atomicity of updates for fields and objects (see the *SCPG* group), and a power-failure during a transaction or the normal runtime does not create the loss of otherwise-permanent data, in the sense that memory on a smart card is essentially persistent with this respect (EEPROM). Data stored on the RAM and subject to such failure is intended to have a limited lifetime anyway (runtime data on the stack, transient objects' contents). According to this, the loss of data within the TSC should be limited to the same restrictions of the transaction mechanism.	**ST]**
[JCSPP 5.1.2] Application note: A package may import at most 128 packages and declare at most 255 classes and interfaces. A class can implement a maximum of 128 public or protected instance methods, and a maximum of 128 instance methods with package visibility. These limits include inherited methods. A class instance can contain a maximum of 255 fields, where an int data type is counted as occupying two fields ([JCVM21], §2.2.4.2).	**[Applet installation not covered by the ST]**
[JCSPP 5.1.3] Application note: The rules described above are strongly inspired in the rules described in section 4.9 of [JVM], *Second Edition*. The complete set of typing rules can be derived from the "Must" clauses from Chapter 7 of [**JCVM21**] as instances of the rules defined above.	[ST 5.3.1.2] Application note:The rules described above are strongly inspired in the rules described in section 4.9 of [JVM], Second Edition. The complete set of typing rules can be derived from the "Must" clauses from Chapter 7 of [**JCVM221**] as instances of the rules defined above.
[JCSPP 5.1.3] Application note: The order relationship between Java Card types is described, for instance, in the description of the checkcast bytecode of [**JCVM21**]. That relation is with the following rules: • Top is the maximum of all types; • Null is the minimum of all classes and array types; • is the minimum of all types. These three extra types are introduced in order to satisfy the two last items in requirement FDP_IFF.2.7.	[ST 5.3.1.2] Application note: The order relationship between Java Card types is described, for instance, in the description of the checkcast bytecode of [**JCVM221**]. That relation is with the following rules: • Top is the maximum of all types; • Null is the minimum of all classes and array types; • ? is the minimum of all types. These three extra types are introduced in order to satisfy the two last items in requirement FDP_IFF.2.7.

Application notes from [JCSPP]	Refined / followed by ST
[JCSPP 5.1.3] Application note: The TYPE attribute of the local variables and the operand stack positions is identified to the attribute of the information they hold. Therefore, this security attribute is possibly modified as information flows. For instance, the rules of the typing function enable information to flow from a local variable *lv* to the operand stack by the operation *sload*, provided that the value of the type attribute of *lv* is *short*. This operation hence modifies the type attribute of the top of the stack. The modification of the security attributes should be done according to the typing rules derived from *Chapter 7 of [**JCVM21**]*.	[ST 5.3.1.3] Application note: The TYPE attribute of the local variables and the operand stack positions is identified to the attribute of the information they hold. Therefore, this security attribute is possibly modified as information flows. For instance, the rules of the typing function enable information to flow from a local variable lv to the operand stack by the operation sload, provided that the value of the type attribute of lv is short. This operation hence modifies the type attribute of the top of the stack. The modification of the security attributes should be done according to the typing rules derived from Chapter 7 of [**JCVM221**].
[JCSPP 5.1.3] Application note: During the type verification of a method, the bytecode verifier makes intensive use of the information provided in the CAP format like the sub-class relationship between the classes declared in the package, the type and class declared for each method and field, the rank of exceptions associated to each method, and so on. All that information can be thought of as security attributes used by the bytecode verifier, or as information relating security attributes. Moreover, the bytecode verifier relies on several properties about the CAP format. All the properties on the CAP format required by the bytecode verifier could, for instance, be completely describedin the TSP model, and the bytecode verifier should ensure that they are satisfied before starting type verifications. Examples of such properties are: • Correspondences between the different components of the CAP file (for instance, each class in the class component has an entry in the descriptor component). • Pointer soundness (example: the index argument in a static method invocation always has an entry in the constant pool); • Absence of hanged pointers (example: each exception handler points to the beginning of some bytecode); • Redundant information (enabling	[ST 5.3.1.4] Application note: During the type verification of a method, the bytecode verifier makes intensive use of the information provided in the CAP format like the sub-class relationship between the classes declared in the package, the type and class declared for each method and field, the rank of exceptions associated to each method, and so on. All that information can be thought of as security attributes used by the bytecode verifier, or as information relating security attributes. Moreover, the bytecode verifier relies on several properties about the CAP format. All the properties on the CAP format required by the bytecode verifier could, for instance, be completely described in the TSP model, and the bytecode verifier should ensure that they are satisfied before starting type verifications. Examples of such properties are: • Correspondences between the different components of the CAP file (for instance, each class in the class component has an entry in the descriptor component). • Pointer soundness (example: the index argument in a static method invocation always has an entry in the constant pool); • Absence of hanged pointers (example: each exception handler points to the beginning of some bytecode); • Redundant information (enabling different ways of searching for it); • Conformance to the Java Language

Application notes from [JCSPP]	Refined / followed by ST
different ways of searching for it); • Conformance to the Java Language Specification respecting the access control features mentioned in §2.2 of [**JCVM22**]. • Packages that are loaded post-issuance can not contain native code.	Specification respecting the access control features mentioned in §2.2 of [**JCVM221**]. • Packages that are loaded post-issuance can not contain native code.
[JCSPP 5.1.3] Application note: The TYPE attribute of the fields and methods is fixed by the application provider and never modified. When a method is invoked, the operand (type) stack is empty. The initial type assigned to those local variables that correspond to the method parameters is the type the application provider declared for those parameters. Any other local variable used in the method is set to the default value Top.	[ST 5.3.1.5] Application note:The TYPE attribute of the fields and methods is fixed by the application provider and never modified. When a method is invoked, the operand (type) stack is empty. The initial type assigned to those local variables that correspond to the method parameters is the type the application provider declared for those parameters. Any other local variable used in the method is set to the default value Top.
[JCSPP 5.1.3] Application note: The intent is to have none of the identified roles to have privileges with regards to the default values of the TYPE attributes.	[ST 5.3.1.5] Application note:The intent is to have none of the identified roles to have privileges with regards to the default values of the TYPE attributes.
[JCSPP 5.1.4] Application note: However, the *S.ADEL* may be granted privileges ([JCRE22], §11.3.5) to bypass the preceding policies. For instance, the logical deletion of an applet renders it un-selectable; this has implications on the management of the associated TSF data (see application note of FMT_MTD.1.1/JCRE).	**[Applet Deletion not covered by the ST]**
[JCSPP 5.1.4] Application note: The modification of the ActiveApplets security attribute should be performed in accordance with the rules given in [JCRE22], §4.	**[Applet Deletion not covered by the ST]**
[JCSPP 5.1.4] Application note: Deleted freed resources (both code and data) may be reused, depending on the way they were deleted (logically or physically). Requirements on *de-allocation* during applet/package deletion are described in [JCRE22], §11.3.4.1, §11.3.4.2 and §11.3.4.3.	**[Applet Deletion not covered by the ST]**
[JCSPP 5.1.4] Application note: There is no conflict with *FDP_ROL.1* requirements appearing in the document as of the bounds on the rollback: the deletion operation is out of the scope of	**[Applet Deletion not covered by the ST]**

Application notes from [JCSPP]	Refined / followed by ST
the rollback (*FDP_ROL.1.1/FIREWALL*, p.73).	
[JCSPP 5.1.4] Application note: The TOE may provide additional feedback information to the card manager in case of a potential security violation (see *FAU_ARP.1*).	**[Applet Deletion not covered by the ST]**
[JCSPP 5.1.5] characterization. The security attributes involved in the rules that determine what a valid remote object reference is are the attribute Returned References of the *O.RMI_SERVICE* and the attribute ActiveApplets (see *FMT_REV.1.1/JCRMI* and *FMT_REV.1.2/JCRMI*).	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([JCRE22], §8.5.2 and [JCAPI22]).	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: Array parameters of remote method invocations must be allocated on the card as global arrays objects. References to global arrays cannot be stored in class variables, instance variables or array components. The control of the flow of that kind of information has already been specified in *FDP_IFC.1.1/JCVM.*	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: The modification of the ActiveApplets security attribute should be performed in accordance with the rules given in [JCRE22], §4.	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: The Exported status of a remote object can be modified by invoking its methods export() and unexport(), and only the owner of the object may perform the invocation without raising a SecurityException (javacard.framework.service. CardRemoteObject). However, even if the owner of the object may provoke the change of the security attribute value, the modification itself could be performed by the JCRE.	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: Remote	**[Remote Method Invocation not covered**

Application notes from [JCSPP]	Refined / followed by ST
objects' security attributes are created and initialized at the creation of the object, and except for the Exported attribute, the values of the attributes are not longer modifiable. The default value of the Exported attribute is true.	**by the ST]**
[JCSPP 5.1.5] Application note: There is one default value for the *SELECTed applet context* that is the *default applet identifier's context*, and one default value for the *active context*, that is "*JCRE*".	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: The intent is to have none of the identified roles to have privileges with regards to the default values of the security attributes. Notice that creation of objects is an operation controlled by the *FIREWALL SFP*; the latitude on the parameters of this operation is described there.	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: The rules previously mentioned are described in [JCRE22], §8.5.	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.5] Application note: applet s own Remote interface objects and may choose to allow or forbid their exportation, which is managed through a security attribute.	**[Remote Method Invocation not covered by the ST]**
[JCSPP 5.1.6] Application note: The modification of the active context, SELECTed applet Context and ActiveApplets security attributes should be performed in accordance with the rules given in [JCRE22], §4 *and ([JCVM22], §3.4.*.	**Same as in [JCSPP 5.1.1.1], covered by [ST 5.1.1.6]**
[JCSPP 5.1.6] Application note: The events that provoke the de-allocation of any transient object are described in [JCRE22], *§5.1.*	**Same as in [JCSPP 5.1.1.2], covered by [ST 5.1.2.6]**
[JCSPP 5.1.6] Application note: The clearing of CLEAR_ON_DESELECT objects is not necessarily performed when the owner of the objects is deselected. In the presence of multiselectable applet instances, CLEAR_ON_DESELECT memory segments may be attached to applet s that are active in different logical channels. Multiselectable applet instances within a same package must share the transient memory segment if	**[Logical channels not covered by the ST]**

Application notes from [JCSPP]	Refined / followed by ST
they are concurrently active ([JCRE22], *§4.2*.	
[JCSPP 5.1.7] Application note: Freed data resources resulting from the invocation of the method javacard.framework.JCSystem. requestObjectDeletion() may be reused. Requirements on *de-allocation* after the invocation of the method are described in [JCAPI22].	**[Object deletion not covered by the ST]**
[JCSPP 5.1.7] Application note: There is no conflict with *FDP_ROL.1* here because of the bounds on the rollback mechanism: the execution of requestObjectDeletion() is not in the scope of the rollback because it must be performed in between APDU command processing, and therefore no transaction can be in progress.	**[Object deletion not covered by the ST]**
[JCSPP 5.1.7] Application note: The TOE may provide additional feedback information to the card manager in case of potential security violation (see *FAU_ARP.1*).	**[Object deletion not covered by the ST]**
[JCSPP 5.1.8] Application note: If this is the case and a new application package is received by the card for installation, the card manager shall first check that it actually comes from the verification authority. The verification authority is the entity responsible for bytecode verification.	**[Secure carrier not covered by the ST]**
[JCSPP 5.1.8] Application note: The exact limitations on the evidence of origin are implementation dependent. In most of the implementations, the card manager performs an immediate verification of the origin of the package using an electronic signature mechanism, and no evidence is kept on the card for future verifications.	**[Object deletion not covered by the ST]**
[JCSPP 5.1.8] Application note: The list of TSF-mediated actions is implementation-dependent, but package installation requires the user to be identified. Here by user is meant the one(s) that in the Security Target shall be associated to the role(s) defined in the component *FMT_SMR.1/CM.*	**[Object deletion not covered by the ST]**
[JCSPP 5.1.8] Application note: The	**[Object deletion not covered by the ST]**

Application notes from [JCSPP]	Refined / followed by ST
security attributes used to enforce the PACKAGE LOADING SFP are implementation dependent. More precisely, they depend on the communication protocol enforced between the CAD and the card. For instance, some of the attributes that can be used are : (1) the keys used by the subjects to encrypt/decrypt their messages; (2) the number of pieces the application package has been split into in order to be sent to the card; (3) the ordinal of each piece in the decomposition of the package, and so on. See for example Appendix D of [GP].	
[JCSPP 5.1.8] Application note: The precise set of rules to be enforced by the function is implementation dependent. The whole exchange of messages shall verify at least the following two rules: (1) the subject S.CRD shall accept a message only if it comes from the subject S.CAD; (2) the subject S.CRD shall accept an application package only if it has received without modification and in the right order all the APDUs sent by the subject S.CAD.	**[Object deletion not covered by the ST]**
[JCSPP 5.1.8] Application note: Modification errors should be understood as modification, substitution, unrecoverable ordering change of data and any other integrity error that may cause the application package to be installed on the card to be different from the one sent by the CAD.	**[Object deletion not covered by the ST]**
[JCSPP 5.1.8] Application note: there is no dynamic package loading on the Java Card platform. New packages can be installed on the card only on demand of the card issuer.	**[Object deletion not covered by the ST]**
[JCSPP 5.1.9] Application note: The abstract machine that underlies the TSF comprises the lower levels of the SCP, that is, the OS and its dedicated native applications and/or APIs (for instance, hardware cryptographic functions/buffers), as well as the IC. Self-test of these components is, as an example, included in [PP0010]. These tests are initiated by the TSF of the SCP itself.	**Not included [ST 5.1.5.1].**

Application notes from [JCSPP]	Refined / followed by ST
[JCSPP 5.1.9] Application note: The use of “security domain” here refers to execution space, and should not be confused with other meanings of security domains.	**Not included [ST 5.1.5.5].**
[JCSPP 5.1.9] Application note: This component supports *OE.SCP.SUPPORT*, which in turn contributes to the secure operation of the TOE, by ensuring that these latter and supporting platform security mechanisms cannot be bypassed.	[ST 5.1.5.6] Application note:This component supports O.SCP.SUPPORT, which in turn contributes to the secure operation of the TOE, by ensuring that these latter and supporting platform security mechanisms cannot be bypassed.
[JCSPP 5.1.9] Application note: This requirement comes from the specification of the Java Card platform but is obviously supported in the implementation by a low-level mechanism of the SCP	[JCSPP 5.1.9] Application note: This requirement comes from the specification of the Java Card platform but is obviously supported in the implementation by a low-level mechanism of the SCP
[JCSPP 5.1.10] Application note: It should be noticed that TSF here refers to the security functions of the environment, rather than security functions of the TOE.	**Not included [ST 5.1.6.1].**
[JCSPP 5.1.10] Application note: The list of TSF-mediated actions depends on the particular card manager security architecture implemented, but typically card content modification requires for the user attempting the modification to be identified. Here by user is meant the one(s) that in the Security Target shall be associated to the role(s) defined in the component *FMT_SMR.1/CMGR*	**Not included [ST 5.1.6.6].**

Table 10: Application notes from [JCSPP]

8 Rationale

8.1 Security Objectives Rationale

8.1.1 Coverage of the Security Objectives

In this section it is proven that the security objectives described in section 4 can be traced for all aspects identified in the TOE-security environment and that they are suited to cover them.

At least one security objective results from each assumption, OSP, and each threat. At least one threat, one OSP or assumption exists for each security objective.

	O.PROTECT_DATA	O.OS_DECEIVE	O.SIDE_CHANNEL	O.FAULT_PROTECT	O.PHYSICAL	O.CARD-MANAGEMENT	O.SHRD_VAR_INTEG	O.SHRD_VAR_CONFID	O.FIREWALL	O.NATIVE	O.OPERATE	O.ALARM	O.RESOURCES	O.REALLOCATION	O.SID	O.SCP.IC	O.SCP.RECOVERY	O.SCP.SUPPORT	O.CIPHER	O.PIN-MNGT	O.KEY-MNGT	O.TRANSACTION	O.RND
T.ACCESS_DATA	x																						
T.OS_OPERATE	x										x												
T.OS_DECEIVE		x																					
T.LEAKAGE			x																				
T.FAULT				x																			
T.PHYSICAL					x											x							
T.CONFID-JCS-DATA T.INTEG-JCS-DATA						x			x		x	x			x		x	x					
T.CONFID-APPLI-DATA						x		x	x		x	x	x		x		x	x	x	x	x	x	
T.INTEG-APPLI-DATA						x	x		x		x	x	x		x		x	x	x	x	x	x	
T.SID.1						x			x						x								
T.SID.2									x		x				x		x	x					
T.NATIVE										x													
T.RESOURCES											x			x			x	x					
T.RND																							x

Table 11: Assignment: threats / OSP – security objectives for the TOE

	OE.IC_ORG	OE.DEV_NOS	OE.DEL_NOS	OE.DLV_DATA	OE.NO-INSTALL	OE.NO-DELETION	OE.VERIFICATION	OE.USE_DIAG	OE.USE_KEY	OE.DLV_PROTECT	OE.TEST_OPERATE	OE.NATIVE
T.DEV_NOS		x										
T.DEV_IC	x											
T.DEL_IC_NOS	x		x									
T.DEL				x								
T.CONFID-JCS-CODE T.INTEG-APPLI-CODE T.INTEG-JCS-CODE							x					
T.CONFID-JCS-DATA T.INTEG-JCS-DATA							x					
T.CONFID-APPLI-DATA							x					
T.INTEG-APPLI-DATA							x					
T.EXE-CODE.1							x					
T.EXE-CODE.2							x					
T.NATIVE							x					
A.DLV_PROTECT										x		
A.TEST_OPERATE				x							x	
A.USE_DIAG								x				
A.USE_KEY									x			
A.NATIVE												x
A-NO-DELETION						x						
A.NO.INSTALL					x							
A.VERIFICATION							x					
OSP.IC_ORG	x											

Table 12: Assignment: threats / assumptions / OSP – security objectives for the environment

Parts of the tables were reproduced from the information contained in sections 7.1 [PP0002], and 6.1.1 [JCSPP]. The justifications given in these sections were slightly adapted and taken over in this ST for completeness reasons.

The following three points must be taken into account when comparing the justification from [PP0002] and [JCSPP] with the justifications given below:

- O.OPERATE was refined to cover additional aspects of threat T.OS.Operate not contained in [JCSPP].
- O.CARD-MANAGEMENT, O.SCP.RECOVERY, O.SCP.SUPPORT, and O.SCP.IC are security objectives for the environment in [JCSPP]. They are independent of the other objectives. So the justifications from [JCSPP] for OE.CARD-MANAGEMENT, OE.SCP.RECOVERY, OE.SCP.SUPPORT, and OE.SCP.IC apply for these objectives.

- T.Physical was refined to cover additional aspects of O.SCP.IC not contained in [JCSPP]. Therefore, the justification from [JCSPP] was adapted.

In the following the justifications are given.

O.PROTECT_DATA addresses the protection of the sensitive information (User Data or TSF data) stored in memories against unauthorized access. The TOE shall ensure that sensitive information stored in memories is protected against unauthorized disclosure and any corruption or unauthorized modification, which covers ***T.ACCESS_DATA*** and modification by unauthorized commands sequence, which covers partially ***T.OS_OPERATE***.

O.SIDE_CHANNEL addresses the protection of the security critical parts of the TOE and protects them from any disclosure by interpretation of physical or logical behavior based on leakage observation (e. g. side channel attacks). Especially, the NOS must be designed to avoid interpretations of electrical signals from the hardware part of the TOE. These characteristics cover the currents, voltages, power consumption, radiation, or timing of signals during the processing activity of the TOE. It allows covering entirely ***T.LEAKAGE***.

O.OPERATE addresses directly the threat ***T.OS_OPERATE*** by ensuring the correct continuation of operation of the TOE logical security functions. Security mechanisms have to be implemented to avoid fraudulent usage of the TOE or usage of incorrect or unauthorized instructions or commands or sequence of commands. The security mechanisms must be designed to always put the TOE in a known and secure state.

O.OS_DECEIVE addresses directly the threat ***T.OS_DECEIVE*** by ensuring that any loss of integrity cannot endanger the security, especially in case of modification of system flags or security attributes (e.g. cryptographic keys). The TOE shall prevent the fraudulent modification of such information as indicators or flags in order to go backwards, through the card life cycle sequence to gain access to prohibited information. The TOE must also guarantee the integrity of the NOS to prevent the modification of its expected behavior (for instance, code patch or rewriting).

O.FAULT_PROTECT ensures the correct continuation of operation of its security functions, in case of interruptions or changes carried out by physical actions (statically or dynamically). The TOE must ensure its correct operation even outside the normal operating conditions where reliability and secure operation has not been proven or tested. This is to prevent errors. The environmental conditions may include voltage, clock frequency, temperature, or external energy fields that can be applied on all interfaces of the TOE (physical or electrical). This addresses directly ***T.FAULT***.

O.PHYSICAL and **O.SCP.IC** ensures protection against physical manipulation of the IC, including the NOS and its application data (TSF data and User data). It prevents from disclose/modify TOE security features or functions provided by the IC. This covers ***T.PHYSICAL***.

OE.DEV_NOS requires the Smart Card NOS to be design in a secure manner and secure environment, ensuring integrity and confidentiality of NOS related information The development tools shall provide for the integrity, availability and reliability of both

programs and data. This specificity will protect against cloning. Information Technology equipment is used to develop, to test, debug, modify, load the OS and personalize the TOE. Therefore, this equipment shall be accessible only by authorized personnel. It must be ensured that confidential information (such as user manuals and general information on defined assets) is only delivered to the parties authorized personnel. This covers threat: ***T.DEV_NOS***.

OE.DEL_NOS addresses the threat applicable to the delivery of the Smart Card embedded Software (Native Operating System and applications) to the IC designer since it requires the application of a trusted delivery and verification procedure to maintain the integrity and the confidentiality of the software if applicable and of initialization data and test information. It must be ensured that Initialization Data are only delivered to the parties authorized personnel and that Initialization Data integrity is achieved. This objective covers the threat ***T.DEL_IC_NOS*** which concerns the NOS developer delivery to IC manufacturer.

OE.IC_ORG requires the IC manufacturer to develop and manufacture the IC in a secure manner, thus responding to the threat ***T.DEV_IC*** to protect confidentiality and integrity of TOE information during phase 2 to 3. This objective also requires covers the threat ***T.DEL_IC_NOS*** during information exchange with NOS developer.

Security procedures required by **OE.DLV_DATA** protect against disclosure or modification Application Data during the delivery to the others manufacturers and thus covers ***T.DEL***.

OE.DLV_PROTECT ensures the following:

- Protection of TOE material/information under delivery and storage,
- Corrective actions are taken in case of improper operation in the delivery process and highlights all non-conformance to this process
- People dealing with the procedure for delivery have got the required skill, training and knowledge to meet the procedure requirements

This objective is directly traced back to ***A.DLV_PROTECT***.

OE.DLV_DATA ensures Card Manufacturer and Personalizer use trusted delivery and verification procedure that are able to maintain the integrity and confidentiality of the TOE sensitive data. This objective is directly traced back to ***A.TEST_OPERATE***.

OE.TEST_OPERATE ensures that appropriate functionality testing of the TOE is used in phases 4 to 6 and security procedures are used to maintain confidentiality and integrity of the TOE. This objective is directly traced back to ***A.TEST_OPERATE***.

OE.USE_DIAG ensures that secure communication protocols and procedures are used between the Smart Card and the terminal during phase 7 and helps to materialize ***A.USE_DIAG***.

OE.USE_KEYS ensures that the keys that stored by terminals or system outside the TOE control are protected in confidentiality. This objective is directly traced back to ***A.USE_KEYS***.

OE.IC_ORG ensures that procedures dealing with physical, personnel, organizational, technical measures for the confidentiality and integrity, of Smart Card Native Operating

System (e.g. source code mask and any associated documents) and IC Manufacturer proprietary information (tools, software, documentation, dice ...) exist and are applied in IC development and manufacturing. This objective correctly covers ***OSP.IC_ORG***.

Justifications from [JCSPP]:

Confidentiality & Integrity

These are generic threats on code and data of *Java Card System* and *applet*s: *T.*CONFID-JCS-CODE, *T.*CONFID-APPLI-DATA, *T.*CONFID-JCS-DATA, *T.*INTEG-APPLI-CODE, *T.*INTEG-JCS-CODE, *T.*INTEG-APPLI-DATA, and *T.*INTEG-JCS-DATA.

Threats concerning the integrity and confidentiality of code are countered by the list of properties described in the (#.*VERIFICATION*) security issue. Bytecode verification ensures that each of the instructions used on the Java Card platform is used for its intended purpose and in the intended scope of visibility. As none of those instructions enables to read or modify a piece of code, no Java Card applet can therefore be executed to disclose or modify a piece of code. Native applications are also harmless because of the objective (O.NATIVE) and the assumption (*A.NATIVE*), so no application can be run to disclose or modify a piece of code.

The (#.*VERIFICATION*) security issue is addressed in this configuration by the objective for the environment *OE.VERIFICATION*.

The threats concerning confidentiality and integrity of data are countered by bytecode verification and the isolation commitments stated in the (O.FIREWALL) objective. This latter objective also relies in its turn on the correct identification of *applets* stated in (O.SID). Moreover, as the firewall is dynamically enforced, it shall never stop operating, as stated in the (O.OPERATE) objective.

As the firewall is a software tool automating critical controls, the objective *O.*ALARM asks for it to provide clear warning and error messages, so that the appropriate counter-measure can be taken.

Concerning the confidentiality and integrity of application sensitive data, as *applet*s may need to share some data or communicate with the *CAD*, cryptographic functions are required to actually protect the exchanged information (*O.*CIPHER). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the *applet*s to use them. Keys and PIN's are particular cases of an application's sensitive data[35] that ask for appropriate management (*O.*KEY-MNGT, *O.*PIN-MNGT, *O.*TRANSACTION). If the PIN class of the Java Card API is used, the objective (O.FIREWALL) is also concerned.

Other application data that is sent to the *applet* as clear text arrives to the *APDU buffer*, which is a resource shared by all applications. The disclosure of such kind of data is prevented by the (*O.SHRD_VAR_CONFID*) security objective. The integrity of the information stored in that buffer is ensured by the (*O.SHRD_VAR_INTEG*) objective.

[35] The *Java Card System* may possess keys as well.

Finally, any attempt to read a piece of information that was previously used by an application but has been logically deleted is countered by the *O.REALLOCATION* objective. That objective states that any information that was formerly stored in a memory block shall be cleared before the block is reused.

Identity Usurpation

T.SID.1 As impersonation is usually the result of successfully disclosing and modifying some assets, this threat is mainly countered by the objectives concerning the isolation of application data (like PINs), ensured by the (O.FIREWALL). Uniqueness of subject-identity (O.SID) also participates to face this threat. Note that the *AIDs*, which are used for *applet* identification, are TSF data.

In this configuration, usurpation of identity resulting from a malicious installation of an applet on the card is covered by the objective *OE.NO-INSTALL*: applets are always installed in a secured environment that prevents any malevolent manipulation of the applets and cards.

T.SID.2 This is covered by integrity of TSF data, subject–identification (O.SID), the *firewall* (O.FIREWALL) and its good working order (O.OPERATE).

Unauthorized Executions

T.EXE-CODE.1 Unauthorized execution of a method is prevented by the objective *OE.VERIFICATION*. This threat particularly concerns the point (8) of the security issue (access modifiers and scope of visibility for classes, fields and methods). The O.FIREWALL objective is also concerned, because it prevents the execution of non-shareable methods of a class instance by any subject apart from the class instance owner.

T.EXE-CODE.2 Unauthorized execution of a method fragment or arbitrary data is prevented by the objective *OE.VERIFICATION*. This threat particularly concerns those points of the security issue related to control flow confinement and the validity of the method references used in the bytecodes.

T.NATIVE An *applet* tries to execute a native method to bypass some security function such as the *firewall*. A Java Card technology-based *applet* (“Java Card applet”) can only access native methods indirectly (O.NATIVE) that is, through an API which is assumed to be secure (*A.NATIVE*). In addition to this, the bytecode verifier also prevents the program counter of an applet to jump into a piece of native code by confining the control flow to the currently executed method (*OE.VERIFICATION*).

Denial of Service

T.RESOURCES	An attacker prevents correct operation of the *Java Card System* through consumption of some resources of the card. This is directly countered by objectives on resource-management (O.RESOURCES) for runtime purposes and good working order (O.OPERATE) in a general manner. Note that, for what relates to CPU usage, the Java Card platform is single–threaded and it is possible for an ill–formed application (either native or not) to monopolize the CPU. However, a smart card can be physically interrupted (card removal or hardware reset) and most CADs implement a timeout policy that prevents them from being blocked should a card fail to answer.

The objective *O.CARD-MANAGEMENT* supports *OE.VERIFICATION* and contributes to cover all the threats on confidentiality and integrity of code and data. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.*SID*.1*.

Finally, the objectives *O.*SCP*.RECOVERY* and *O.*SCP*.SUPPORT* are intended to support the O.OPERATE, *O.*ALARM and O.RESOURCES objectives of the TOE, so they are indirectly related to the threats that these latter objectives contribute to counter.

The objective *OE.NATIVE* ensures that the environmental assumption *A.NATIVE* is upheld. The objective *OE.VERIFICATION* upholds the assumption *A.VERIFICATION*.

The assumptions *A.NO-DELETION* and *A.NO-INSTALL* are also upheld by the objective *O.CARD-MANAGEMENT*.

The following security objectives of the TOE are related to the assumptions made for this configuration as follows:

O.FIREWALL	The controlled sharing of data owned by different applications assumes that the code of the applications is well typed (*A.VERIFICATION*). Secured installation ensures the correct initialization of TSF data such as the identity of the applications (*A.NO-INSTALL*).
O.SID	The correct identification of the applications depends on the assumptions stating that pre-issuance applications have been correctly installed (*A.NO-INSTALL*), and that those are exactly the applications that will be on the card (*A.NO-DELETION*).

Justification from [PP0002]:

O.RND covers ***T.RND*** because the objective is are stated in a way, which directly corresponds to the description of the threat. It is clear from the description of the objective, that the corresponding threat is removed if the objective is valid. More specifically, in every case the ability to use the attack method successfully is countered, if the objective holds.

8.2 Security Requirements Rationale

8.2.1 Security Functional Requirements Rationale

This section proves that the quantity of security requirements (TOE and environment) is suited to fulfill the security objectives described in section 4 and that it can be traced back to the security objectives.

8.2.1.1 TOE Security Requirements Rationale

All security objectives of the TOE are met by the security functional requirements. At least one security objective exists for each security functional requirement.

	O.PROTECT_DATA	O.SIDE_CHANNEL	O.OS_DECEIVE	O.FAULT_PROTECT	O.PHYSICAL	O.CARD-MANAGEMENT	O.SHRD_VAR_INTEG	O.SHRD_VAR_CONFID	O.FIREWALL	O.NATIVE	O.OPERATE	O.ALARM	O.RESOURCES	O.REALLOCATION	O.SID	O.SCP.IC	O.SCP.RECOVERY	O.SCP.SUPPORT	O.CIPHER	O.PIN-MNGT	O.KEY-MNGT	O.TRANSACTION	O.RND
FAU																							
ARP.1											x	x	x										
SAA.1			x	x							x												
FCS																							
CKM.1																			x		x		
CKM.2																			x		x		
CKM.3	x	x																	x		x		
CKM.4	x	x																	x		x		
COP.1 (all iterations)	x	x																	x		x		
RND.1																							x
FDP																							
ACC.1/CMGR						x																	
ACC.2/FIREWALL	x								x		x									x			
ACF.1/FIREWALL ACF.1/CMGR	x					x			x		x									x			
ETC.1	x										x												
IFC.1/JCVM							x	x	x														
IFF.1/JCVM							x	x	x														
ITC.1	x										x												
RIP.1 (all iterations)	x							x			x			x						x	x	x	
ROL.1/FIREWALL																				x		x	
SDI.2	x										x									x	x		
FIA																							
AFL.1/PIN AFL.1/CMGR	x					x					x												
ATD.1	x										x				x								
UAU.1	x										x												
UAU.3	x										x												
UAU.4	x										x												
UID.1/CMGR	x					x					x												
UID.2															x								

	O.PROTECT_DATA	O.SIDE_CHANNEL	O.OS_DECEIVE	O.FAULT_PROTECT	O.PHYSICAL	O.CARD-MANAGEMENT	O.SHRD_VAR_INTEG	O.SHRD_VAR_CONFID	O.FIREWALL	O.NATIVE	O.OPERATE	O.ALARM	O.RESOURCES	O.REALLOCATION	O.SID	O.SCP.IC	O.SCP.RECOVERY	O.SCP.SUPPORT	O.CIPHER	O.PIN-MNGT	O.KEY-MNGT	O.TRANSACTION	O.RND
USB.1	x										x				x								
FMT																							
LIM.1	x																						
LIM.2	x																						
MSA.1/JCRE MSA.1/CMGR			x			x			x						x								
MSA.2/JCRE			x						x														
MSA.3/FIREWALL MSA.3/CMGR			x			x			x						x								
MTD.1/JCRE	x		x						x				x		x								
MTD.3									x				x		x								
SMF.1	x		x			x			x		x		x										
SMR.1/JCRE SMR.1/CMGR	x					x			x		x		x										
FPR																							
UNO.1		x									x								x	x	x		
FPT																							
AMT.1/SCP											x							x					
EMSEC.1		x																					
FLS.1/SCP				x							x	x	x				x						
PHP.1					x											x							
PHP.3/SCP					x											x							
RCV.3/SCP																	x	x					
RCV.4/SCP																		x					
RVM.1									x	x	x		x		x			x					
SEP.1	x	x	x	x					x		x				x			x					
TDC.1											x												
TST.1				x							x												
FRU																							
FLT.2/SCP																	x						
FTP																							
ITC.1						x																	

Table 13: Assignment: TOE security requirements – TOE security objectives

The explanations given in the full Security Target were not intended to be published and have therefore been removed from this Security Target Lite.

8.2.1.2 IT-Environment Security Requirements Rationale

The security objectives that address the IT environment are OE.VERIFICATION, OE.USE_DIAG, and OE.USE_KEYS. OE.VERIFICATION is met by all security functional requirements from the group BCVG (section 5.3.1): FDP_IFC.2, FDP_IFF.2, FMT_MSA.1, FMT_MSA.2, FMT_MSA.3, FMT_SMR.1 and FRU_RSA.1 as justified in section 6.2.1.2 of

[JCSPP] and OE.USE_DIAG and OE.USE_KEYS are met by FTP_ITC.1/ENV. All other security objectives for the environment address non-IT aspects only. FMT_SMF.1 has been defined in addition to [JCSPP] but for compliance to [CC] and FI065 only. FMT_SMF.1/BCV is mapped to the same OE as FMT_MSA.1/BCV to OE.VERIFICATION.

8.2.1.3 Fulfilling all Dependencies

The set of security functional requirements that are selected covers all the TOE security objectives as demonstrated in section 8.2.1.2.

The following table identifies all ST security functional requirements (TOE and IT environment) and their associated dependencies. It also indicates whether the ST explicitly addresses each dependency. For those cases where dependencies have not specifically been addressed, explanations of the rationale for excluding them are provided.

No.	Class / Component	Dependency	Dependency satisfied
	FAU		
1.	FAU_ARP.1/JCS	FAU_SAA.1	No. 2
2.	FAU_SAA.1	FAU_GEN.1	no, see below
	FCS		
3.	FCS_CKM.1	[FCS_CKM.2 or FCS_COP.1] FCS_CKM.4, FMT_MSA.2	No. 4 No. 6, 38
4.	FCS_CKM.2	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2	No. 3 No. 6, 38
5.	FCS_CKM.3	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2	No. 3 No. 6, 38
6.	FCS_CKM.4	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FMT_MSA.2	No. 3 No. 38
7.	FCS_COP.1 (all iterations)	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1] FCS_CKM.4, FMT_MSA.2	No. 3 No. 6, 38
8.	FCS_EMSEC.1	no	-
9.	FCS_RND.1	no	-
	FDP		
10.	FDP_ACC.1/CMGR	FDP_ACF.1	No. 12
11.	FDP_ACC.2/FIREWALL	FDP_ACF.1	No. 13
12.	FDP_ACF.1/CMGR	FDP_ACC.1, FMT_MSA.3	No. 10, 40

No.	Class / Component	Dependency	Dependency satisfied
13.	FDP_ACF.1/FIREWALL	FDP_ACC.1, FMT_MSA.3	No. 11, 41
14.	FDP_ETC.1	[FDP_ACC.1 or FDP_IFC.1]	No. 10
15.	FDP_IFC.1/JCVM	FDP_IFF.1	No. 17
16.	FDP_IFC.2/BCV	FDP_IFF.1	No. 18
17.	FDP_IFF.1/JCVM	FDP_IFC.1, FMT_MSA.3	No. 15, 41
18.	FDP_IFF.2/BCV	FDP_IFC.1, FMT_MSA.3	No. 16, 42
19.	FDP_ITC.1	[FDP_ACC.1 or FDP_IFC.1] FMT_MSA.3	No. 10 No. 40
20.	FDP_RIP.1 (all iterations)	no	-
21.	FDP_ROL.1/FIREWALL	[FDP_ACC.1 or FDP_IFC.1]	No. 11
22.	FDP_SDI.2	no	-
	FIA		
23.	FIA_AFL.1/CMGR	FIA_UAU.1	No. 26
24.	FIA_AFL.1/PIN	FIA_UAU.1	No. 26
25.	FIA_ATD.1/AID	no	-
26.	FIA_UAU.1	FIA_UID.1	No. 29
27.	FIA_UAU.3/CMGR	no	-
28.	FIA_UAU.4/CMGR	no	-
29.	FIA_UID.1/CMGR	no	-
30.	FIA_UID.2/AID	no	-
31.	FIA_USB.1	FIA_ATD.1	No. 25
	FMT		
32.	FMT_LIM.1	FMT_LIM.2	No. 33
33.	FMT_LIM.2	FMT_LIM.1	No. 32
34.	FMT_MSA.1/CMGR	[FDP_ACC.1 or FDP_IFC.1] FMT_SMF.1, FMT_SMR.1	No. 10 No. 47, 48
35.	FMT_MSA.1/JCRE	[FDP_ACC.1 or FDP_IFC.1] FMT_SMF.1, FMT_SMR.1	No. 11 No. 46, 49
36.	FMT_MSA.1/BCV.1	[FDP_ACC.1 or FDP_IFC.1] FMT_SMF.1, FMT_SMR.1	No. 16 No. 45, 50
37.	FMT_MSA.1/BCV.2	[FDP_ACC.1 or FDP_IFC.1] FMT_SMF.1, FMT_SMR.1	No. 16 No. 45, 50

No.	Class / Component	Dependency	Dependency satisfied
38.	FMT_MSA.2/JCRE	ADV_SPM.1 [FDP_ACC.1 or FDP_IFC.1] FMT_MSA.1, FMT_SMR.1	EAL4 No. 11 No. 35, 49
39.	FMT_MSA.2/BCV	ADV_SPM.1 [FDP_ACC.1 or FDP_IFC.1] FMT_MSA.1, FMT_SMR.1	EAL4 No. 16 No. 36, 37, 50
40.	FMT_MSA.3/CMGR	FMT_MSA.1, FMT_SMR.1	No. 32, 48
41.	FMT_MSA.3/JCRE	FMT_MSA.1, FMT_SMR.1	No. 35, 49
42.	FMT_MSA.3/BCV	FMT_MSA.1, FMT_SMR.1	No. 36, 37, 50
43.	FMT_MTD.1/JCRE	FMT_SMF.1, FMT_SMR.1	No. 46, 49
44.	FMT_MTD.3	ADV_SPM.1, FMT_MTD.1	EAL4, No. 43
45.	FMT_SMF.1/BCV	no	-
46.	FMT_SMF.1/JCRE	no	-
47.	FMT_SMF.1/CMGR	no	-
48.	FMT_SMR.1/CMGR	FIA_UID.1	No. 29
49.	FMT_SMR.1/JCRE	FIA_UID.1	No. 30
50.	FMT_SMR.1/BCV	FIA_UID.1	no, see below
	FPR		
51.	FPR_UNO.1	no	-
	FPT		
52.	FPT_AMT.1/SCP	no	-
53.	FPT_FLS.1/JCS	ADV_SPM.1	EAL4
54.	FPT_FLS.1/SCP	ADV_SPM.1	EAL4
55.	FPT_PHP.1	no	-
56.	FPT_PHP.3/SCP	no	-
57.	FPT_RCV.3/SCP	AGD_ADM.1 and ADV_SPM.1	EAL4
58.	FPT_RCV.4/SCP	ADV_SPM.1	EAL4
59.	FPT_RVM.1	no	-
60.	FPT_RVM.1/SCP	no	-
61.	FPT_SEP.1	no	-
62.	FPT_SEP.1/SCP	no	-
63.	FPT_TDC.1	no	-
64.	FPT_TST.1	FPT_AMT.1	No. 52
	FRU		

No.	Class / Component	Dependency	Dependency satisfied
65.	FRU_FLT.2/SCP	FPT_FLS.1	No. 54
66.	FRU_RSA.1/BCV	no	-
	FTP		
67.	FTP_ITC.1/CMGR	no	-
68.	FTP_ITC.1/ENV	no	-

Table 14: Security functional requirement dependencies

The following dependencies are not fulfilled:

1. From FAU_SAA.1 to FAU_GEN.1

   The dependency of FAU_SAA.1 with FAU_GEN.1 is not applicable to the TOE; the FAU_GEN.1 component forces many security relevant events to be recorded (due to dependencies with other functional security components) and this is not achievable in a Smart Card since many of these events result in card being in an insecure state where recording of the event itself could cause a security breach. It is then assumed that the function FAU_SAA.1 may still be used and the specific audited events will have to be defined in the ST independently with FAU_GEN.1.

2. From FMT_SMR.1/BCV to FIA_UID.1

   The following rationale has been taken from section 6.2.1.3 of [JCSPP]:

   FIA_UID.1 is required by the component FMT_SMR.1 in group BCVG. However, the role bytecode verifier defined in this component is attached to an IT security function rather than to a “user” of the CC terminology. The bytecode verifier does not “identify” itself with respect to the TOE. Furthermore, it is part of the IT environment. Thus, here it is claimed that this dependency can be left out.

8.2.1.4 Suitability of Minimum Strength of Function (SOF) Level

The security functions based on permutational or probalistic algorithms are **SOF-high** claimed. This level is required to defeat attackers possessing high attack potential (AVA_VLA.4).

8.2.2 Assurance Requirements Rationale

The assurance requirements of this Security Target are defined by the level EAL4 augmented by ADV_IMP.2, ALC_DVS.2, AVA_VLA.4 and AVA_MSU.3.

8.2.2.1 Evaluation Assurance Level Rationale

An assurance requirement of EAL4 is required for this type of TOE since it is intended to defend against sophisticated attacks. This evaluation assurance level was selected since it is designed to permit a developer to gain maximum assurance from positive security engineering based on good commercial practices. EAL4 represents the highest practical level of assurance expected for a commercial grade product.

In order to provide a meaningful level of assurance that the TOE provides an adequate level of defense against such attacks, the evaluators should have access to the low level design and source code. The lowest for which such access is required is EAL4.

The assurance level EAL4 is achievable, since it requires no specialist techniques on the part of the developer.

8.2.2.2 Assurance Augmentations Rationale

Additional assurance requirements are also required due to the definition of the TOE and the intended security level to assure.

8.2.2.2.1 ADV_IMP.2 Implementation of the TSF

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement. NOS source code is an example of implementation representation. This augmentation is also suitable to capture the detailed internal working of the TSF and especially the generation of random numbers that meet class K3 of [AIS 20] with SOF-high.

This assurance component is a higher hierarchical component to EAL4 (only ADV_IMP.1 is found in EAL4.) It is important for a Smart Card that the evaluator evaluates the implementation representation of the entire TSF to determine if the functional requirements in the Security Target are addressed by the representation of the TSF.

ADV_IMP.2 has dependencies with ADV_LLD.1 “Descriptive Low-Level design”, ADV_RCR.1 “Informal correspondence demonstration”, ALC_TAT.1 “Well defined development tools”. These components are included in EAL4, and so these dependencies are satisfied.

8.2.2.2.2 ALC_DVS.2 Sufficiency of Security Measures

Development security is concerned with physical, procedural, personnel and other technical measures that may be used in the development environment to protect the TOE.

This assurance component is a higher hierarchical component to EAL4 (only ALC_DVS.1 is found in EAL4). Due to the nature of the TOE, there is a need to justify the sufficiency of these procedures to protect the confidentiality and the integrity of the TOE.

ALC_DVS.2 has no dependencies.

8.2.2.2.3 AVA_VLA.4 Highly Resistant

Vulnerability analysis is an assessment to determine whether vulnerabilities identified, during the evaluation of the construction and anticipated operation of the TOE or by other methods (e.g. by flaw hypotheses), could allow users to violate the TSP.

An assurance component that is hierarchically higher to EAL4 (only AVA_VLA.2 is found in EAL4) was chosen to provide the necessary level of protection against attackers with high attack potential.

AVA_VLA.4 has dependencies ADV_FSP.1 “Informal functional specification” (satisfied by ADV_FSP.2, which is included in EAL4), ADV_HLD.2 “Security enforcing high-level design” (included in EAL4 and thus satisfied), ADV_IMP.1 “Subset of the implementation of the TSF” (satisfied by ADV_IMP.2 which is required by this ST), ADV_LLD.1 (included in EAL4 and thus satisfied), AGD_ADM.1 (included in EAL4 and thus satisfied) and AGD_USR.1 (included in EAL4 and thus satisfied).

8.2.2.2.4 AVA_MSU.3 Analysis and Testing for insecure States

Misuse investigates whether the TOE can be configured or used in a manner that is insecure but that an administrator or user of the TOE would reasonably believe to be secure.

This assurance component is a higher hierarchical component to EAL4 (only AVA_MSU.2 is found in EAL4). Due to the nature of the TOE and the required level of protection, there is a need to provide analysis, which can be validated and confirmed through testing by an evaluator.

AVA_MSU.3 has dependencies with ADO_IGS.1 “Installation, generation, and start-up procedures” (included in EAL4 and thus satisfied), ADV_FSP.1 (satisfied by ADV_FSP.2, which is included in EAL4), AGD_ADM.1 (included in EAL4 and thus satisfied) and AGD_USR.1 (included in EAL4 and thus satisfied).

8.2.2.3 Dependencies and Mutual Support

The purpose of this part of the PP rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given in respect to the assurance requirement because:

- EAL4 is an established set of mutually supportive and internally consistent assurance requirements.
- The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies have been satisfied).
- The dependencies analysis for the functional requirements described above, demonstrate mutual support and internal consistency between the functional requirements.
- Inconsistency between functional and assurance requirements can only arise if there are functional-assurance dependencies which are not met, a possibility which has been shown not to arise in the above section "Security functional requirements dependencies".

Additionally:

- On all occasions where different IT security requirements apply to the same types of events, operations, data, tests to be performed etc., these requirements do not conflict and therefore, an appropriate justification is not applicable.

Therefore, the dependencies analysis described above demonstrates mutual support and internal consistency between the functional requirements.

8.3 TOE Summary Specification Rationale

8.3.1 Security Functions Rationale

In this section it is shown that the security functions are suited to fulfill the security requirements. It is demonstrated that at least one security function meets each security requirement. Furthermore it is shown that all security functions are needed and that they form an integrated unity to meet the security requirements.

8.3.1.1 Fulfilling the Security Functional Requirements

The following table provides a mapping of security functions to security functional requirements and is followed by a discussion of how each security functional requirement is addressed by the corresponding security function. A number entry in this table means, that the SFR is covered by the corresponding aspect of the security function. Cross entries occur only for SF.Hardware and mean that the TSF from the hardware as defined in [ST0348] cover the corresponding SFR.

	SF.AccessControl	SF.Audit	SF.CryptoKey	SF.CryptoOperation	SF.I&A	SF.SecureManagement	SF.PIN	SF.Transaction	SF.Hardware
FAU									
FAU_ARP.1/JCS		1-13							
FAU_SAA.1		1-13							
FCS									
FCS_CKM.1			1-3						
FCS_CKM.2			4-6						
FCS_CKM.3			7						
FCS_CKM.4			8						
FCS_COP.1/TripleDES …/RSACipher				1 2					2, 5

	SF.AccessControl	SF.Audit	SF.CryptoKey	SF.CryptoOperation	SF.I&A	SF.SecureManagement	SF.PIN	SF.Transaction	SF.Hardware
…/DESMAC				3					
…/AES				4					3, 5
…/RSASignatureISO9796				5					
…/RSASignaturePKCS#1				6					
…/SHA_1				7					
FCS_RND.1				8					1, 5
FDP									
FDP_ACC.1/CMGR	1								
FDP_ACC.2/FIREWALL	2								
FDP_ACF.1/CMGR	1								
FDP_ACF.1/FIREWALL	2								
FDP_ETC.1	1								
FDP_IFC.1/JCVM	3								
FDP_IFF.1/JCVM	3								
FDP_ITC.1	1								
FDP_RIP.1/OBJECTS						9			
FDP_RIP.1/APDU						9			
FDP_RIP.1/bArray						9			
FDP_RIP.1/TRANSIENT						9			
FDP_RIP.1/ABORT						9			
FDP_RIP.1/KEYS						9			
FDP_ROL.1/FIREWALL								1	
FDP_SDI.2						8			
FIA									
FIA_AFL.1/CMGR					2				
FIA_AFL.1/PIN							2-3		
FIA_ATD.1/AID						5			
FIA_UAU.1							1,4		
FIA_UAU.3/CMGR					1				
FIA_UAU.4/CMGR					1				

	SF.AccessControl	SF.Audit	SF.CryptoKey	SF.CryptoOperation	SF.I&A	SF.SecureManagement	SF.PIN	SF.Transaction	SF.Hardware
FIA_UID.1/CMGR					3				
FIA_UID.2/AID						5			
FIA_USB.1						5			
FMT									
FMT_LIM.1									7
FMT_LIM.2									7
FMT_MSA.1/CMGR FMT_MSA.1/JCRE	4 5								
FMT_MSA.2/JCRE	6								
FMT_MSA.3/CMGR FMT_MSA.3/FIREWALL	7 7								
FMT_MTD.1/JCRE	5								
FMT_MTD.3	6								
FMT_SMF.1	4-6								
FMT_SMR.1/CMGR FMT_SMR.1/JCRE	4-6 4-6								
FPR									
FPR_UNO.1						7			
FPT									
FPT_AMT.1/SCP		1, 2							
FPT_EMSEC.1						10			6
FPT_FLS.1/JCS		1-13							
FPT_FLS.1/SCP						6			4, 5
FPT_PHP.1		2				6			5
FPT_PHP.3/SCP		2							5
FPT_RCV.3/SCP						8		1	
FPT_RCV.4/SCP								1	
FPT_RVM.1 FPT_RVM.1/SCP						2 2			
FPT_SEP.1						1			

	SF.AccessControl	SF.Audit	SF.CryptoKey	SF.CryptoOperation	SF.I&A	SF.SecureManagement	SF.PIN	SF.Transaction	SF.Hardware
FPT_SEP.1/SCP									4,5,7
FPT_TDC.1						11			
FPT_TST.1						6			
FRU									
FRU_FLT.2/SCP									4, 5
FTP									
FTP_ITC.1/CMGR	1								

Table 15: Mapping of functional requirements to security functions

The explanations given in the full Security Target were not intended to be published and have therefore been removed from this Security Target Lite.

8.3.2 Assurance Measures Rationale

Assurance measures and rationale are described in section 6.3.

8.4 Definition of additional Families

8.4.1 Definition of Family FCS_RND

This section has been taken over from the certified (BSI-PP-0002) Smartcard IC Platform Protection profile [PP0002].

Family behavior

This family defines quality requirements for the generation of random numbers which are intended to be use for cryptographic purposes.

Component leveling:

FCS_RND Generation of random numbers	1

FCS_RND.1 Generation of random numbers requires that random numbers meet a defined quality metric.

Management: FCS_RND.1

There are no management activities foreseen.

Audit: FCS_RND.1

There are no actions defined to be auditable.

FCS_RND.1 Quality metric for random numbers

Hierarchical to: No other components.

FCS_RND.1.1 The TSF shall provide a mechanism to generate random numbers that meet [assignment: a defined quality metric].

Dependencies: No dependencies.

8.4.2 Definition of the Family FPT_EMSEC

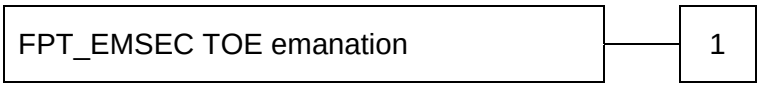
This section has been taken over from the certified (BSI-PP-0017) Protection Profile *Machine Readable travel Document with "ICAO Application", Basic Access Control* [PP0017].

The additional family FPT_EMSEC (TOE Emanation) of the Class FPT (Protection of the TSF) is defined here to describe the IT security functional requirements of the TOE. The TOE shall prevent attacks against the private signature key and other secret data where the attack is based on external observable physical phenomena of the TOE. Examples of such attacks are evaluation of TOE's electromagnetic radiation, simple power analysis (SPA), differential power analysis (DPA), timing attacks, etc. This family describes the functional requirements for the limitation of intelligible emanations which are not directly addressed by any other component of [CC] part 2.

Family behavior

This family defines requirements to mitigate intelligible emanations.

Component leveling:



FPT_EMSEC.1 TOE emanation has two constituents:

FPT_EMSEC.1.1 Limit of emissions requires to not emit intelligible emissions enabling access to TSF data or user data.

FPT_EMSEC.1.2 Interface emanation requires not emit interface emanation enabling access to TSF data or user data.

Management: FPT_EMSEC.1

There are no management activities foreseen.

Audit: FPT_EMSEC.1

There are no actions defined to be auditable.

FPT_EMSEC.1 TOE Emanation

Hierarchical to: No other components.

FPT_EMSEC.1.1 The TOE shall not emit [assignment: types of emissions] in excess of [assignment: specified limits] enabling access to [assignment: list of types of TSF data] and [assignment: list of types of user data].

FPT_EMSEC.1.2 The TSF shall ensure [assignment: type of users] are unable to use the following interface [assignment: type of connection] to gain access to [assignment: list of types of TSF data] and [assignment: list of types of user data].

Dependencies: No other components.

8.4.3 Definition of the Family FMT_LIM

This section has been taken over from the certified (BSI-PP-0017) Protection Profile *Machine Readable travel Document with "ICAO Application", Basic Access Control* [PP0017].

The family FMT_LIM describes the functional requirements for the Test Features of the TOE. The new functional requirements were defined in the class FMT because this class addresses the management of functions of the TSF. The examples of the technical mechanism used in the TOE show that no other class is appropriate to address the specific issues of preventing the abuse of functions by limiting the capabilities of the functions and by limiting their availability.

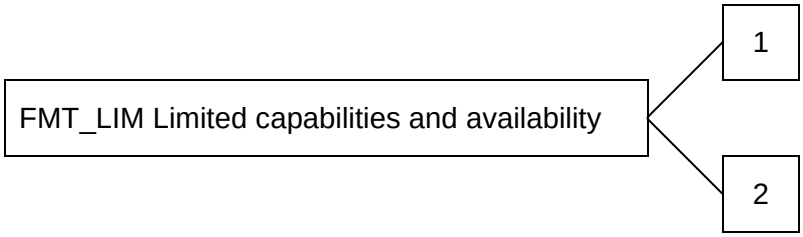
The family "Limited capabilities and availability (FMT_LIM)" is specified as follows.

FMT_LIM Limited capabilities and availability

Family behaviour

This family defines requirements that limit the capabilities and availability of functions in a combined manner. Note that FDP_ACF restricts the access to functions whereas the Limited capability of this family requires the functions themselves to be designed in a specific manner.

Component leveling:



FMT_LIM.1	Limited capabilities requires that the TSF is built to provide only the capabilities (perform action, gather information) necessary for its genuine purpose.
FMT_LIM.2	Limited availability requires that the TSF restrict the use of functions (refer to Limited capabilities (FMT_LIM.1)). This can be achieved, for instance, by removing or by disabling functions in a specific phase of the TOE's life-cycle.
Management:	FMT_LIM.1, FMT_LIM.2 There are no management activities foreseen.
Audit:	FMT_LIM.1, FMT_LIM.2 There are no actions defined to be auditable.

To define the IT security functional requirements of the TOE an additional family (FMT_LIM) of the Class FMT (Security Management) is defined here. This family describes the functional requirements for the Test Features of the TOE. The new functional requirements were defined in the class FMT because this class addresses the management of functions of the TSF. The examples of the technical mechanism used in the TOE show that no other class is appropriate to address the specific issues of preventing the abuse of functions by limiting the capabilities of the functions and by limiting their availability.

The TOE Functional Requirement "Limited capabilities (FMT_LIM.1)" is specified as follows.

FMT_LIM.1	Limited capabilities
Hierarchical to:	No other components.
FMT_LIM.1.1	The TSF shall be designed in a manner that limits their capabilities so that in conjunction with "Limited availability (FMT_LIM.2)" the following policy is enforced [assignment: Limited capability and availability policy].
Dependencies:	FMT_LIM.2 Limited availability.

The TOE Functional Requirement "Limited availability (FMT_LIM.2)" is specified as follows.

FMT_LIM.2	Limited availability.
Hierarchical to:	No other components.
FMT_LIM.2.1	The TSF shall be designed in a manner that limits their availability so that in conjunction with "Limited capabilities (FMT_LIM.1)" the following policy is enforced [assignment: Limited capability and availability policy].
Dependencies:	FMT_LIM.1 Limited capabilities.
Application note:	The functional requirements FMT_LIM.1 and FMT_LIM.2 assume that there are two types of mechanisms (limited capabilities and

limited availability) which together shall provide protection in order to enforce the policy. This also allows that

(i) the TSF is provided without restrictions in the product in its user environment but its capabilities are so limited that the policy is enforced

or conversely

(ii) the TSF is designed with high functionality but is removed or disabled in the product in its user environment.

The combination of both requirements shall enforce the policy.

9 Annex

9.1 Glossary and Abbreviations

9.1.1 Abbreviations

A.xxx	Assumptions
BSI	"*Bundesamt für Sicherheit in der Informationstechnik*", German national certification body
CC	Common Criteria
CM	Card Manger
DCSSI	"*Direction Centrale de la Sécurité des Systèmes d'Information*", French national certification body
EAL	Evaluation Assurance Level
EEPROM	Electrically Erasable Programmable ROM
ES	Embedded Software
HAL	Hardware Abstraction Layer
IC	Integrated Circuit
NOS	Native Operating System. For this ST, NOS means the TOE without the underlying hardware platform, i. e. NOS is equivalent to the smart card embedded software
OSP.xxx	Organizational security policies
O.xxx	Security objectives for the TOE
OE.xxx	Security objectives for the environment
PP	Protection Profile
RAM	Random Access Memory
ROM	Read Only Memory
RTE	Runtime Environment
SC	Smart Card
SF.xxx	Security function
ST	Security Target
SOF	Strength Of Function
T.xxx	Threats
TOE	Target of Evaluation
TSF	TOE Security Functions
VM	Virtual Machine

9.1.2 Glossary

AID — Application identifier, an ISO-7816 data format used for unique identification of Java Card applications (and certain kinds of files in card file systems). The Java Card platform uses the *AID* data format to *identify* **applet**s *and package*s. *AIDs* are administered by the International Standards Organization (ISO), so they can be used as unique identifiers.

AIDs are also used in the security policies (see "*Context*" below): applets' *AIDs* are related to the selection mechanisms, *package*s' *AIDs* are used in the enforcement of the *firewall*. **Note:** although they serve different purposes, they share the same name space.

APDU — Application Protocol Data Unit, an ISO 7816-4 defined communication format between the card and the off-card applications. Cards receive requests for service from the CAD in the form of *APDU*s. These are encapsulated in Java Card System by the **javacard.framework.APDU** *class* ([JCAPI221]).

*APDU*s manage both the selection-cycle of the *applets* (through *JCRE* mediation) and the communication with the *Currently selected applet*.

APDU buffer — The APDU buffer is the buffer where the messages sent (received) by the card depart from (arrive to). The *JCRE* owns an *APDU* object (which is a *JCRE Entry Point* and an instance of the **javacard.framework.APDU** *class*) that encapsulates *APDU* messages in an internal byte array, called the *APDU buffer*. This object is made accessible to the *currently selected applet* when needed, but any permanent access (out-of selection-scope) is strictly prohibited for security reasons.

applet — The name is given to a Java Card technology-based user application. An applet is the basic piece of code that can be selected for execution from outside the card. Each applet on the card is uniquely identified by its *AID*.

applet deletion manager — The on-card component that embodies the mechanisms necessary to delete an applet or library and its associated data on smart cards using Java Card technology.

BCV — The bytecode verifier is the software component performing a static analysis of the code to be loaded on the card. It checks several kinds of properties, like the correct format of *CAP file*s and the enforcement of the typing rules associated to bytecodes. If the component is placed outside the card, in a secure environment, then it is called an off-card verifier. If the component is part of the embedded software of the card it is called an on-card verifier.

CAD Card Acceptance Device, or card reader. The device where the card is inserted, and which is used to communicate with the card.

CAP file A file in the Converted applet format. A CAP file contains a binary representation of a *package* of *class*es that can be installed on a device and used to execute the *package*'s *class*es on a Java Card virtual machine. A CAP file can contain a user library, or the code of one or more applets.

Class In object-oriented programming languages, a class is a prototype for an object. A class may also be considered as a set of objects that share a common structure and behavior. Each class declares a collection of fields and methods associated to its instances. The contents of the fields determine the internal state of a class instance, and the methods the operations that can be applied to it. Classes are ordered within a class hierarchy. A class declared as a specialization (a subclass) of another class (its super class) inherits all the fields and methods of the latter.

Java platform classes should not be confused with the classes of the functional requirements (FIA) defined in the CC.

Context A context is an object-space partition associated to a *package*. Applets within the same Java technology-based *package* belong to the same context. The *firewall* is the boundary between contexts (see "*Current context*").

Current context The *JCRE* keeps track of the current Java Card System context (also called "the active context"). When a virtual method is invoked on an object, and a context switch is required and permitted, the current context is changed to correspond to the context of the *applet* that owns the object. When that method returns, the previous context is restored. Invocations of static methods have no effect on the current context. The current context and sharing status of an object together determine if access to an object is permissible.

Currently selected applet The applet has been selected for execution in the current session. The *JCRE* keeps track of the currently selected Java Card applet. Upon receiving a SELECT command from the *CAD* with this applet's *AID*, the *JCRE* makes this applet the currently selected applet. The *JCRE* sends all *APDU* commands to the currently selected applet ([JCRE221] Glossary).

Default applet The applet that is selected after a card reset ([JCRE221], §4.1).

Embedded Software Pre-issuance loaded software.

Firewall The mechanism in the Java Card technology for ensuring *applet* isolation and object sharing. The firewall prevents an applet in

one *context* from unauthorized access to objects owned by the *JCRE* or by an applet in another context.

Installer

The installer is the on-card application responsible for the installation of applets on the card. It may perform (or delegate) mandatory security checks according to the card issuer policy (for bytecode-verification, for instance), loads and link *package*s (*CAP file*(s)) on the card to a suitable form for the *JCVM* to execute the code they contain. It is a subsystem of what is usually called “card manager”; as such, it can be seen as the portion of the card manager that belongs to the TOE.

The installer has an *AID* that uniquely identifies him, and may be implemented as a Java Card applet. However, it is granted specific privileges on an implementation-specific manner ([JCRE221], §10).

Interface

A special kind of Java programming language *class*, which declares methods, but provides no implementation for them. A class may be declared as being the implementation of an interface, and in this case must contain an implementation for each of the methods declared by the interface. (see also *shareable interface*).

JCRE

The Java Card runtime environment consists of the Java Card virtual machine, the Java Card API, and its associated native methods. This notion concerns all those dynamic features that are specific to the execution of a Java program in a smart card, like *applet* lifetime, applet isolation and object sharing, transient objects, the transaction mechanism, and so on.

JCRE Entry Point

An object owned by the *JCRE* context but accessible by any application. These methods are the gateways through which applets request privileged *JCRE* system services: the instance methods associated to those objects may be invoked from any context, and when that occurs, a context switch to the *JCRE* context is performed.

There are two categories of JCRE Entry Point Objects: Temporary ones and Permanent ones. As part of the *firewall* functionality, the *JCRE* detects and restricts attempts to store references to these objects.

JCRMI

Java Card Remote Method Invocation is the Java Card System, version 2.2.2, mechanism enabling a client application running on the *CAD* platform to invoke a method on a remote object on the card. Notice that in Java Card System, version 2.1.1, the only method that may be invoked from the CAD is the **process** method of the **applet** class.

Java Card System

The Java Card System: the *JCRE* (*JCVM* +API), the *installer*, and the on-card *BCV* (if the configuration includes one).

JCVM	The embedded interpreter of bytecodes. The JCVM is the component that enforces separation between applications (*firewall*) and enables secure data sharing.
logical channel	A logical link to an application on the card. A new feature of the Java Card System, version 2.2.2, that enables the opening of up to four simultaneous sessions with the card, one per logical channel. Commands issued to a specific logical channel are forwarded to the active applet on that logical channel.
Object deletion	The Java Card System, version 2.2.2, mechanism ensures that any unreferenced persistent (transient) object owned by the current context is deleted. The associated memory space is recovered for reuse prior to the next card reset.
Package	A *package* is a name space within the Java programming language that may contain *class*es and *interface*s. A *package* defines either a user library, or one or more applet definitions. A *package* is divided in two sets of files: export files (which exclusively contain the public *interface* information for an entire *package* of *class*es, for external linking purposes; export files are not used directly in a Java Card virtual machine) and *CAP file*s.
SCP	Smart Card Platform. It is comprised of the integrated circuit, the operating system and the dedicated software of the smart card.
Shareable interface	An interface declaring a collection of methods that an *applet* accepts to share with other applets. These *interface* methods can be invoked from an *applet* in a *context* different from the context of the object implementing the methods, thus “traversing” the *firewall*.
SIO	An object of a class implementing a *shareable interface*.
Subject	An active entity within the TOE that causes information to flow among objects or change the system’s status. It usually acts on the behalf of a user. Objects can be active and thus are also *subjects* of the TOE.
Transient object	An object whose contents is not preserved across CAD sessions. The contents of these objects are cleared at the end of the current CAD session or when a card reset is performed. Writes to the fields of a transient object are not affected by transactions.
User	Any application interpretable by the *JCRE*. That also covers the *package*s. The associated subject(s), if applicable, is (are) an object(s) belonging to the **javacard.framework.applet** *class*.

9.2 References

[AGD_ADM] Administrator Guide NXP P541G072V0P (JCOP 41 v2.3.1) Secure Smart Card Controller, NXP P531G072V0P (JCOP 31 v2.3.1) Secure Smart Card Controller, NXP P531G072V0Q (JCOP 31 v2.3.1) Secure Smart Card Controller, and NXP P521G072V0P (JCOP 21 v2.3.1) Secure Smart Card Controller, Version 1.8, 2007-06-01, IBM Deutschland Entwicklung GmbH

[AGD_USR] User Guidance NXP P541G072V0P (JCOP 41 v2.3.1) Secure Smart Card Controller, NXP P531G072V0P (JCOP 31 v2.3.1) Secure Smart Card Controller, NXP P531G072V0Q (JCOP 31 v2.3.1) Secure Smart Card Controller, and NXP P521G072V0P (JCOP 21 v2.3.1) Secure Smart Card Controller, Version 1.3, 2007-06-01, IBM Deutschland Entwicklung GmbH

[AIS 20] Anwendungshinweise und Interpretationen zum Schema, AIS 20: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren, Version 1, 02.12.1999, Bundesamt für Sicherheit in der Informationstechnik

[AIS 31] Anwendungshinweise und Interpretationen zum Schema, AIS 31: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren, Version 1, 25.09.2001, Bundesamt für Sicherheit in der Informationstechnik

[ANSI X9.62] ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American National Standards Institute

[BioAPI] Biometric Application Programming Interface (API) for Java Card, NIST/Biometric Consortium, Biometric Interoperability, Assurance, and Performance Working Group, Version 1.1, 7 August 2002

[BSI0348] Certificate BSI-DSZ-CC-0348-2006 as of date March 2006 NXP P5CT072V0P Secure Smart Card Controller from Philips Semiconductor GmbH – Business Line Identification

[CC] Common Criteria for Information Technology Security Evaluation, version 2.3, August 2005
Part 1: Introduction and general model, CCMB-2005-08-001,
Part 2: Security functional requirements, CCMB-2005-08-002,
Part 3: Security Assurance Requirements, CCMB-2005-08-003.

[CSRS] GlobalPlatform Card Security Requirements Specification, Version 1.0, May 2003.

[FIPS 46-3] FIPS PUB 46-3: FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, DATA ENCRYPTION STANDARD (DES), Reaffirmed 1999 October 25, U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

[FIPS 180-1]. FIPS PUB 180-1: FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, SECURE HASH STANDARD, 1995 April 17

[FIPS 197]. FIPS PUB 197: Federal Information Processing Standards Publication 197, Announcing the ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001

[GP] GlobalPlatform Card Specification, Version 2.1.1, March 2003.

[ISO 9796-2] ISO/IEC 9796-2:2002: Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms

[ISO 9797-1] ISO/IEC 9797-1:1999: Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher

[JAVASPEC] The Java Language Specification. Gosling, Joy and Steele. ISBN 0-201-63451-1.

[JCAPI21] Java Card 2.1.1 Application Programming Interface. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCAPI22] Java Card 2.2 Application Programming Interface. June 2002. Published by Sun Microsystems, Inc.

[JCAPI221] Java Card 2.2.1 Application Programming Interface. October 21, 2003. Published by Sun Microsystems, Inc.

[JCBV] Java Card 2.1.2 Off-Card Verifier. January 2001. White paper. Published by Sun Microsystems, Inc.

[JCRE21] Java Card 2.1.1 Runtime Environment (JCRE) Specification. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCRE22] Java Card 2.2 Runtime Environment (JCRE) Specification. June 2002. Published by Sun Microsystems, Inc.

[JCRE221] Java Card 2.2.1 Runtime Environment (JCRE) Specification. October 2003. Published by Sun Microsystems, Inc.

[JCSPP] Java Card System Protection Profile Collection, Version: 1.0b, August 2003.
This Document contains 4 protection profile, whereas “Java Card System – Minimal Configuration Protection Profile” (registered at DCSSI under Registration number PP/0303) is relevant for this ST.

[JCVM21] Java Card 2.1.1 Virtual Machine (JCVM) Specification. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCVM22] Java Card 2.2 Virtual Machine (JCVM) Specification. June 2002. Published by Sun Microsystems, Inc.

[JCVM221] Java Card 2.2.1 Virtual Machine (JCVM) Specification. October 2003. Published by Sun Microsystems, Inc.

[JVM] The Java Virtual Machine Specification. Lindholm, Yellin. ISBN 0-201-43294-3.

[OPCS]	Open Platform Card Specification, Version 2.0.1', 7 April 2000
[OPPP]	Open Platform Protection Profile (OP3), Version 0.7, 31 January, 2001
[PKCS#1 v1.5]	PKCS #1: RSA Encryption Standard – An RSA Laboratories Technical Note, Version 1.5, Revised November 1, 1993
[PP0002]	Smartcard IC Platform Protection Profile, Version 1.0, July 2001 (registered at BSI under Registration number BSI-PP-0002)
[PP0017]	Common Criteria Protection Profile – Machine Readable Travel Document with "ICAO Application", Basic Access Control, Version 1.0, 18.08.2005 (registered at BSI under Registration number BSI-PP-0017)
[ST0348]	Security Target Lite, BSI-DSZ-CC-0348, Version 1.2, 17.01.2006, Evaluation of the Philips P5CT072V0P, P5CC072V0P, P5CD072V0P and P5CD036V0P Secure Smart Card Controllers