Security Target Lite ProxSIM Taurus

Version 1.0/ Status 13.05.2011

Giesecke & Devrient GmbH
Prinzregentenstr. 159
Postfach 80 07 29
D-81607 München

Contents

1 ST Introduction

1.1 ST Reference

Title: Security Target Lite ProxSIM Taurus
Reference: ProxSIM Taurus_ASE

Version Number: Version 1.0/ Status 13.05.2011

Origin: Giesecke & Devrient GmbH

CC Version: 3.1 (Revision 3)

Assurance Level: EAL4-augmented with the following assurance components: ALC_DVS.2 and AVA_VAN.5.

PP Claim: Demonstrable conformace to: [JCSPP].

TOE: ProxSIM Taurus

Version of the TOE: 1.0

TOE documentation as pdf-document or print-out version :

- User Guidance Main Document ProxSIM Taurus, V. 1.4, 20.04.2011
- User Guidance ProxSIM Taurus, V. 1.6, 20.04.2011
- Application Development in User Phase ProxSIM Taurus, V. 1.4, 04.04.2011
- User Guidance Initialisation ProxSIM Taurus, V. 1.6, 20.04.2011

HW-Part of TOE: S3FS91J [STL], certified according to Common Criteria Version 3.1, ANSSI-CC-2009/57.

1.2 ST Overview

The aim of this document is to describe the Security Target for **ProxSIM Taurus.** In the following chapters **ProxSIM Taurus** stands for the Target of Evaluation (TOE).

The related product is the **ProxSIM Taurus OS Java Card**.
In the following chapters, ProxSIM Taurus Java Card stands for the product.

ProxSIM Taurus Java Card contains the TOE (see Figure 1 green line) consisting of the:

- JCRE, JCVM, Java Card API´s, Remote Method Invocation (RMI), logical channels and applet and object deletion,
- the native telecommunication related application

- and the SCP (Smart Card Platform) consisting of IC (Integrated Circuit), OS (Chip Operating System) and DS (Chip Dedicated Software)

and depends on the secure IT environment consisting of the off card Byte Code Verification.

Part of the ProxSIM Taurus OS Java Card is a fully interoperable GlobalPlatform [GP22] compliant multi-application Java Card OS.

ProxSIM Taurus consists of the related software in combination with the underlying hardware ('Composite Evaluation').

This Security Target claims demonstrable conformance to the: [JCSPP].

This document describes:

- the Target of Evaluation (TOE): ProxSIM Taurus
- the security environment of the TOE: Security Problem Definition: chapter 4
- the security objectives of the TOE and its environment: chapter 5
- the TOE security functional and assurance requirements: chapter 6
- the TOE summary specification: chapter 7

The assurance level for the TOE is CC **EAL4 augmented**.

1.2.1 Sections Overview

Section 1 provides the introductory material for the Security Target.

Section 2 provides the conformance claim of the Security Target.

Section 3 provides the general security aspects of the Security Target.

Section 4 provides a discussion of the expected environment for the TOE. This section also defines the set of threats that are to be addressed by either the technical countermeasures implemented in the TOE hardware, the TOE software, or through the environmental controls.

Section 5 defines the security objectives for both the TOE and the TOE environment.

Section 6 contains the functional requirements and assurance requirements derived from the Common Criteria [CC1], Part 2 [CC2] and Part 3 [CC3], which must be satisfied.

Section 7 contains the TOE Summary Specification.

Section 8 provides a consitency mapping to the threats, objectives and security functional and security assurance requirements offered by the chip certification.

Section 9 provides information on used definition and acronyms.

Section 10 provides a information on references used.

1.3 Typographic Conventions

- *This typeface* is used to highlight those words that appear in the Definitions. Example: *applet*.
- **This typeface** is used to highlight assignments and selections for SFRs completed by the ST author.
- **This typeface** is used to highlight assignments and selections for SFRs defined in the PP.

1.4 Change History

Version	Date	Changes
1.0	13.05.2011	Final version of the Security Target Lite. Generated from version 3.4 of the Security Target.

1.5 Figures

1.6 Tables

1.7 Application Notes of the PP

When applicable the application notes of the PP are discussed in notes.

Some application notes of the PP are taken into account but are not explicitly discussed because they are either only important for the better understanding or are trivial.

1.8 TOE Overview

The TOE type is the Java Card System (Java Card RE, Java Card VM and Java Card API[1]) compliant with Java Card specifications versions 2.2.2 on top of a Basic OS from G&D and embedded in an already certified Integrated Circuit (IC). The native telecommunication related application is also part of the TOE. The card manager is not part of the TOE, but is always delivered together with the TOE on the same smart card. The TOE facilitates post issuance loading of applications. The TOE is integrated in a mobile phone solution and provides services for SIM-based mobile NFC.

NFC services as public transport ticketing, payment, loyalty and event ticketing might be executed by the TOE.

For public transport, tickets or tokens are stored in the NFC application and users can request access to the transportation system by swiping their mobile. A mobile NFC payment transaction is achieved by swiping the mobile over an NFC reader at a point of sale.

The TOE provides the following interfaces to the mobile phone:

- The ISO-interface according to [ISO7816-3] and [ISO7816-4] and the
- SWP-interface according ETSI.

The major TOE security features are implemented in the Java Card System and are supported by the underlying SCP (G&D Basic OS and the IC). The SCP provides support in case of memory management functions, I/O functions, transaction facilities and secure (shielded, native) implementation of cryptographic functions. The major TOE security features are:

- The Installer, which is responsible for:
  - Secure Loading, to download a CAP-file to the smart card.
  - Secure Linking, to speed up the execution of the application. Linking includes a resolution and a preparation step.
  - Secure Installation of the applet on the card by using an application identifier (AID).
  - Secure Deletion of applets: Applet instance deletion, applet/library package deletion and deletion of an applet package and contained instances.
- The Java Card Virtual Machine (JCVM), which is the bytecode interpreter.

[1] Please note, that the Java Card API includes the GP API. This will be valid for the complete document.

- The Java Card RE, which is responsible for parts of the card resource management, communication, applet execution and applet security.
- The Java Card API, that provides classes and interfaces to the Java Card applets. It defines the calling conventions by which an applet may access the Java Card RE and native services provided by the SCP such as, I/O management functions, PIN and cryptographic specific management and the exceptions mechanism.
- The Java Card Firewall. In the Java Card platform, applet isolation is achieved through the applet firewall mechanism, which is also part of the TOE security features. However applet isolation cannot be entirely 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, which is off-card and is not part of the TOE security features, but part of the TOE-environment.
- The Java Card System Remote Method Invocation (JCRMI), which supports logical channels.

The following Non-TOE part is required to be used:

- The off-card byte code verifier, which has to be applied to all CAP-files that will be loaded onto the TOE.

1.9 TOE Description

This section presents the architecture and the common usages of the Target of Evaluation (TOE) in more detail than already described in chapter 1.8.

1.9.1 TOE Type

The TOE type is the Java Card System (Java Card RE including the Java Card VM, the Java Card API [2]and the Installer) on top of a G&D Basic OS embedded in Integrated Circuit (IC) from Samsung, compliant with Java Card specifications versions 2.2.2, and with post-issuance application downloading facilities. The native telecommunication related application and its APIs including the STK is also part of the TOE.

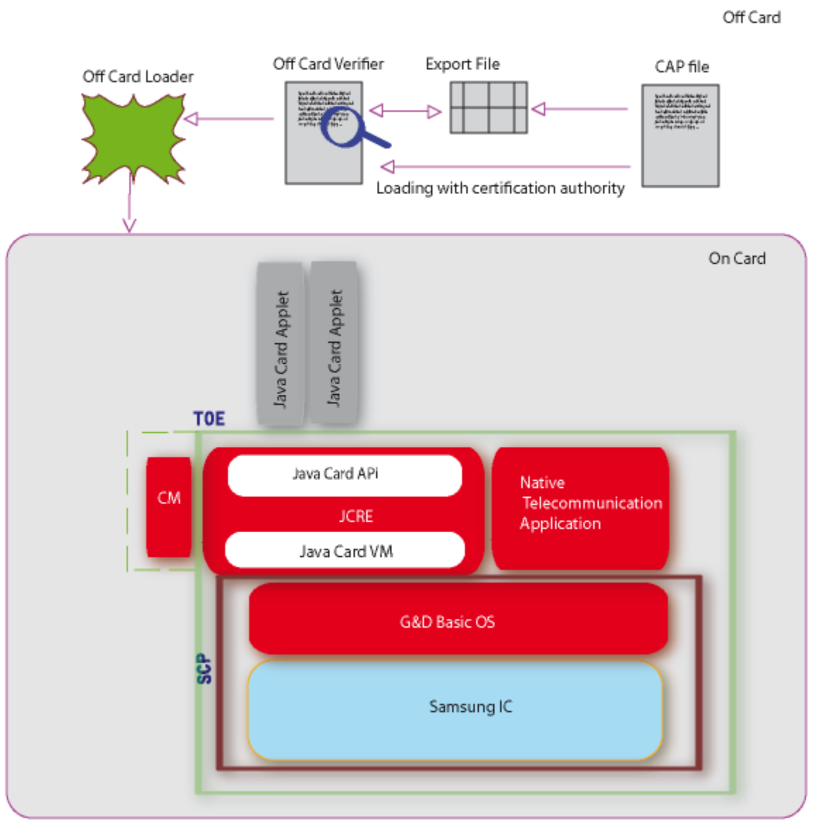

Figure 1 The TOE and its environment (red parts including white boxes are parts from G&D). The Card Manager is always delivered together with the TOE and marked by a dashed line.

[2] Please note, that the Java Card API includes the GP API. This will be valid for the complete document.

The TOE comprises:

1. Java Card System’s code and data stored in the IC memories.
2. The native telecommunication related application.
3. The Smart Card Platform including
   a. the already certified IC and
   b. the Basic OS from Giesecke & Devrient.
4. Guidance delivered to the Card Issuer.

The Card Manager will also always be delivered together with the TOE.

The TOE form factor that will be delivered is a mask version ready for initialization and personalization of the final smartcard for end customers. It comprises all G&D-software (see – red parts) running on top of the hardware.

Note, that the certification of the IC covers more security features than necessary and used by the G&D Basic OS and the Java Card System.

Unless stated otherwise, in the rest of the PP, TOE stands for the Java Card System and the SCP without guidance.

1.9.2 TOE Security Features

The Java Card System Open Configuration considered in [JCSPP] implements Java Card Specifications version 2.2.2 ([JCRE222], [JCVM222], [JCAPI222]) and allows postissuance downloading of applications that have been previously verified by an off-card trusted IT component. Figure 1 places the different components of the TOE, the Java Card System and the SCP, in their environment.

Application handling in the TOE-environment:

The development of the applets is carried on in a Java programming environment. The compilation of the code produces the corresponding class file. Then all class files of the package is processed by the converter[3], which validates the code and generates a converted CAP-file, the equivalent of a Java™ package for the Java Card platform. A CAP file contains an executable binary representation of the classes of a package. A package is a namespace within the Java programming language that may contain classes and interfaces, and in the context of Java Card technology, it defines either a user

[3] The converter is defined in the specifications [JCVM221] as the off-card component of the Java Card virtul machine.

library, or one or several applets. Then, the off-card bytecode verifier checks the CAP file.

Application handling by the TOE:

After the validation is carried out, the CAP file is loaded into the card by means of a safe loading mechanism.

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 by using a type of cryptographic certification (Secure Channel Protocol = 02, see [GP22]). Once the identification step is accomplished, the CAP file is transmitted to the card. Due to resource limitations, usually the file is split by the card issuer into a list of Application Protocol Data Units (APDUs), which are in turn sent to the card. Authentication of the external entity, loading and initialisation are parts of the TOE security features.

Once loaded into the card the file is linked, what makes it possible in turn to install, if defined, instances of any of the applets defined in the file.

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.

Both an export file and a CAP file contain the major and minor version numbers of the package described. When a CAP file is installed on a Java Card technology-enabled device a resident image of the package is created, and the major and minor version numbers are recorded as part of that image. When an export file is used during preparation of a CAP file, the version numbers indicated in the export file are recorded in the CAP file.
During installation, references from the package of the CAP file being installed to an imported package can be resolved only when the version numbers indicated in the export file used during preparation of the CAP file are compatible with the version numbers of the resident image. They are compatible when the major version numbers are equal and the minor version of the export file is less than or equal to the minor version of the resident image. After that we have two further steps:

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 [JVM]. In the next step, called in [JVM] the preparation step, the static field image[4] 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.

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 applet instances within the card. In particular, the AID is used for selecting the applet instance for execution. In some cases, the actual installation (and registration) of applets is

[4] The memory area containing the statis fields of the file.

postponed; in the same vein, a package may contain several applets, and some of them might never be installed.

Installation is separated from the process of loading and linking a CAP file on the card.

The installer is the Java Card System component dealing with secure loading of CAP files, linking and installation of new packages, as described in [JCRE222]. 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. The installer is also in charge of applet deletion ([JCRE222], §11.3.4):

- Applet instance deletion, which is the removal of the applet instance and the objects owned by the applet instance.
- Applet/library package deletion, which entails the removal of all the card resident components of the CAP file, including code and any associated JCRE management structures.
- Deletion of an applet package and contained instances, which is the removal of the card resident code and JCRE structures associated with the applet package, and all the applet instances in the context of the package.

The Java Card VM is the bytecode interpreter as specified in [JCVM22]. The Java Card RE is responsible for parts of the card resource management, communication, applet execution and applet security. The Java Card API provides classes and interfaces to the Java Card applets. It defines the calling conventions by which an applet may access the Java Card RE and native services provided by the SCP such as, I/O management functions, PIN and cryptographic specific management and the exceptions mechanism.

While the Java Card VM is responsible for ensuring language-level security, the Java Card RE provides additional security features for Java Card technology-enabled devices. Applets 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 ([JCRE222] §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 applets, unless a “shareable interface” is explicitly provided (by the applet who owns it) for allowing access to that information. The Java Card RE allows sharing using the concept of “shareable interface objects” (SIO) and static public variables. Java Card VM dynamically enforces the firewall, that is, at runtime. However applet isolation cannot be entirely 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 Java Card VM ensures that the only way for applets to access any resources are either through the Java Card RE or through the Java Card API or APIs of the

Telecommunication Native Application including the STK. This objective can only be guaranteed if applets are correctly typed (all the “must clauses” imposed in chapter 7 of [JCVM22] on the bytecodes and the correctness of the CAP file format are satisfied).

The Java Card System compliant with Java Card specification versions 2.2.2 also implements the Java Card System Remote Method Invocation (JCRMI) and supports logical channels.

JCRMI provides a mechanism for a client application running on the CAD platform to invoke a method on a remote object on the card. The CAD issues commands to the card, which in turn dispatches them to the appropriate object. The applet owner of those objects controls the access to exported objects and the JCRE ensures coherence and synchronization of the remote object with its on-card representative.

Logical channels allow a terminal to open multiple sessions into the smart card, one session per logical channel ([JCRE222], §4). Commands may be issued on a logical channel to instruct the card either to open or to close a logical channel. An applet instance that is selected to be active on a channel shall process all the commands issued to that channel. The platform also introduces the possibility for an applet instance to be selected on multiple logical channels at the same time, or accepting other applets belonging to the same package to be selected simultaneously. These applets are referred to as multiselectable. A non-multiselectable applet can be active at most on one channel. Applets within a package are either all multiselectable or all non-multiselectable.

The Java Card System provides:

- Object deletion upon request of an applet instance. The JCRE ensures that any unreferenced object owned by that instance is deleted and the associated space is recovered for reuse.

Below the Java Card System the SCP (Basic G&D OS and the IC) provides support in case of memory management functions, I/O functions, transaction facilities and secure (shielded, native) implementation of cryptographic functions. The following cryptographic functions will be supported by the TOE:

- RSA for
  - signature generation / verification ([PKCS1], [RFC2409]) and
  - for encryption / decryption ([ISO14888], [PKCS1], [JCAPI222]).
- AES for
  - encryption / decryption ([AES], [JCAPI222])
  - CBC-MAC ([JCAPI222])
- 3-DES for
  - encryption / decryption ([ISO9797], [JCAPI222], [PKCS5])
  - CBC-MAC ([JCAPI222], [ISO9797], [PKCS5])

The interfaces of the TOE that could be used are:

- The ISO-interface (ISO 7816-3) with transmission protocol T=0 according to [ISO7816-3] and [ISO7816-4].
- The SWP-interface according to ETSI.

The native telecommunication application is part of the TOE, but does not provide security features to the TOE in the sense of the SFRs.

1.9.3 Non-TOE HW/SW/FW available to the TOE

The following sections further describe the components involved in the environment of the Java Card System. The role they play will help in understanding the importance of the assumptions on the environment of the TOE.

1.9.3.1 Bytecode Verification

The bytecode verifier is a program that performs static checks on the bytecodes of the methods of a CAP file prior to the execution of the file on the card. 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 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. Bytecodeverification could 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 implementationdependent, but it is required that it should at least enforce all the “must clauses” imposed in [JCVM22] on the bytecodes and the correctness of the CAP files’ format.

1.9.3.2 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 Java Card RE. The card manager is in charge of the life cycle of the whole card, as well as the installed applications (applets). 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) or the proximity-coupling device (PCD)[5]. It is the controller of the card, but relies on the TOE to manage the runtime of client applets. In this TOE the Global Platform card manager [GP], will be used.

1.9.4 TOE Life Cycle

The G&D software life cycle is embedded in the final product life cycle, i.e. the Java Card platform with applications, that goes from product development to its usage by the final user. The product life cycle phases are those detailed in Figure 2. We refer to [PP0035] for a thorough description of Phases 1 to 7:

- Phases 1 and 2 compose the product development: Embedded Software (IC Dedicated Software, OS, Java Card System, other platform components such as Card Manager, Applets) and IC development.
- Phase 3 and Phase 4 correspond to IC manufacturing and packaging, respectively. Some IC pre-personalisation steps may occur in Phase 3.
- Phase 5 concerns the embedding of software components within the IC (Initialisation).
- Phase 6 is dedicated to the product personalisation prior final use.
- Phase 7 is the product operational phase.

The life cycle of the G&D software is composed of four stages:

- Development,
- Storage, pre-personalisation and testing
- Personalisation and testing
- Final usage.

G&D software storage is not necessarily a single step in the life cycle since it can be stored in parts. G&D software delivery occurs before storage. These stages map to the typical smartcard life cycle phases as shown in Figure 2.

[5] The acronym CAD is used here and throughout this specification to refer to both types of card readers - the conventional Card Acceptance Device (CAD) for contacted I/O interfaces and the Proximity Coupling Device (PCD) for contactless interfaces.

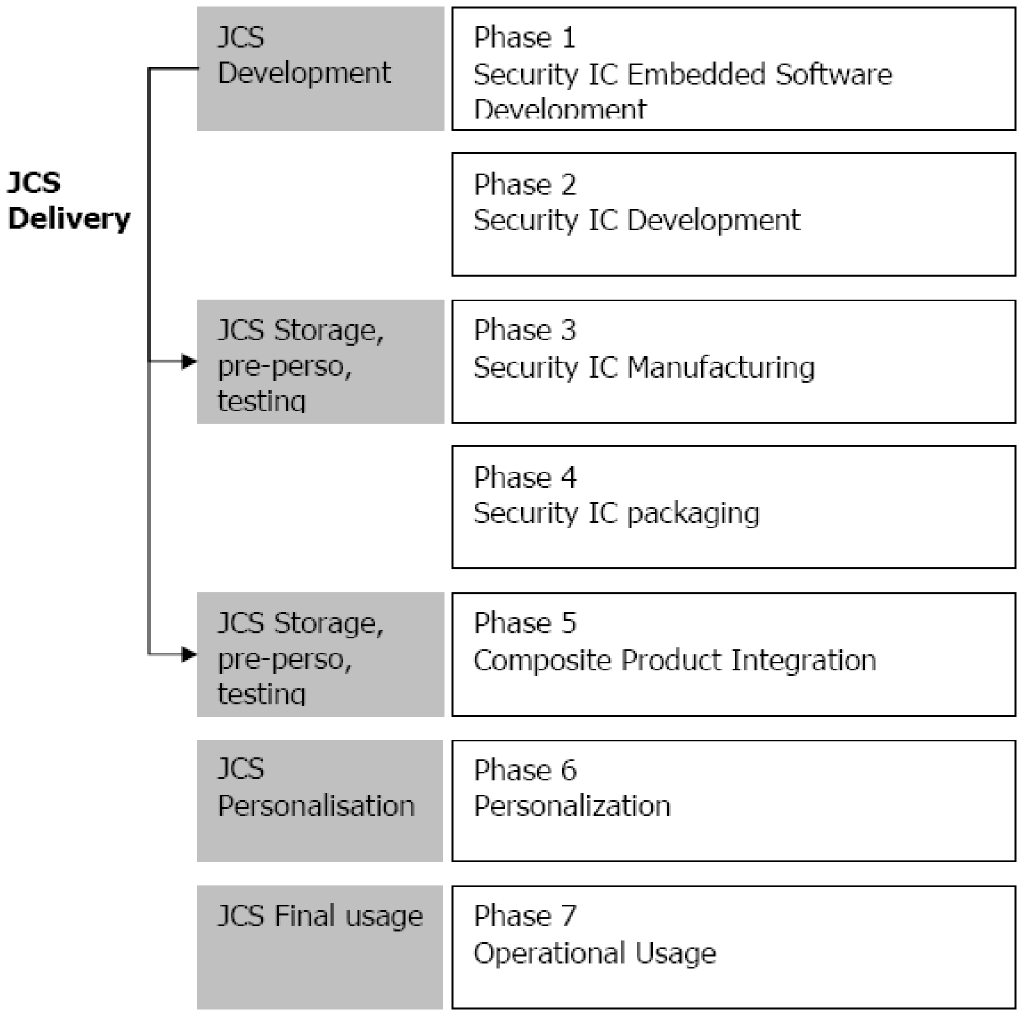
JCS Delivery
JCS Development
Phase 1 Security IC Embedded Software Development
Phase 2 Security IC Development
JCS Storage, pre-perso, testing
Phase 3 Security IC Manufacturing
Phase 4 Security IC packaging
JCS Storage, pre-perso, testing
Phase 5 Composite Product Integration
JCS Personalisation
Phase 6 Personalization
JCS Final usage
Phase 7 Operational Usage

Figure 2 G&D software Life Cycle within Product Life Cycle . Together with the JCS other software-parts of the TOE will be delivered.

G&D software Development is performed during Phase 1. This includes JCS conception, design, implementation, testing and documentation. The JCS as part of the G&D software development shall fulfill requirements of the final product, including conformance to Java Card Specifications, and recommendations of the SCP user guidance. The G&D software development shall occur in a controlled environment that avoids disclosure of source code, data and any critical documentation and that guarantees the integrity of these elements. The evaluation of a product against this PP will include the G&D software development environment.

The delivery of the G&D software will occur during Composite Product Integration (Phase 5). Delivery and acceptance procedures shall guarantee the authenticity, the confidentiality and integrity of the exchanged pieces. G&D software delivery shall usually involve encrypted signed sending and it supposes the previous exchange of public keys. The evaluation includes the delivery process.

The evaluation of a product against the PP shall include the whole Security IC Manufacturing environment. As the Security IC has already been certified there is no need to perform the evaluation again.

In Phase 5, the Composite Product Integrator stores and pre-personalize the G&D software and potentially conducts tests on behalf of the G&D software developer. Therefore the Composite Product Integrator performs the initialization process. In this phase the executable code is loaded. The Composite Product Integration environment shall protect the integrity and confidentiality of the G&D software and of any related material, for instance test suites. Note that JCS storage in Phase 5 implies a product delivery after Phase 5 to the personalization facility.

The TOE will be personalized in Phase 6. In this phase card individual data are loaded (e.g. Card Manager keys, personalised applet data). Phase 6 is already part of the usage phase.

The product shall be tested again and all critical material including personalization data, test suites and documentation shall be protected from disclosure and modification.

The G&D software final usage environment is that of the product where the JCS is embedded in. It covers a wide spectrum of situations that cannot be covered by evaluations. The JCS and the product shall provide the full set of security functionalities to avoid abuse of the product by untrusted entities.

The point of TOE-delivery is when a mask version ready for initialisation (the Master Init) and the certified HW-platform S3FS91J will be delivered for final initialization and personalization. This Master Init loaded on the HW-Platform S3FS91J represents the TOE. As a step during initialization the Master Init could be used to generate a Maxi Init.

After personalization (life-cycle phase 6) the TOE is in its evaluated configuration and therefore secured. During initialization and personalization phase the user should follow the related guidance documentation.

Note 1: The product is a flash product. The JCS will be developed at G&D and loaded onto the HW at the beginning of phase 5. No JCS-related storing will be performed in phase 3. The JCS will be delivered in advance to phase 5. For further advise please read the user guidances.

1.9.5 TOE Usage

Smart cards are used as data carriers that are secure against forgery and tampering as well 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, 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 communicates with a card reader.

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, or the balance of an electronic purse.

So far, the most typical 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) or the NFC chip for mobile phones.
- 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.

2 Conformance claims

2.1 CC conformance Claim

This Security Target claims to be conformant to the Common Criteria version 3.1, which comprises:

- Common Criteria for Information Technology Security Evaluation, Part 1: [CC1],
- Common Criteria for Information Technology Security Evaluation, Part 2: [CC2],
- Common Criteria for Information Technology Security Evaluation, Part 3: [CC3],

It claims to be CC Part 2 extended and CC Part 3 conformant. The extension to CC Part 2 are extensions made in the Platform certification of the Samsung chip and are defined in [STL].

The

- Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, [CEM]

has to be taken into account.

2.2 PP Claim

This ST is demonstrable compliant with the Java Card ^{TM} System Protection Profile, [JCSPP].

2.3 PP Additions and Refinements

The SCP is part of the TOE in this ST, while the SCP is not part of the TOE in [JCSPP]. The following changes have been performed to cover the extension of the TOE:

The following objectives for the operational environment: OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.SCP.IC have been moved from [JCSPP] to objectives for the TOE in this ST. A new objective for the operational environment has been introduced: OE.SYM_KEY_GEN to support secure symmetric key generation.

No additional SFRs have been introduced.

2.4 Package Claim

The current ST is conformant to the following security requirements package:

- Assurance package EAL4 augmented with ALC_DVS.2 and AVA_VAN.5 as defined in the CC, part 3 [CC3].

2.5 Conformance Claim Rationale

2.5.1 TOE Type

The TOE type stated in chapter 1.1 is commensurate with the current TOE type in the claimed PP [JCSPP] as only the SCP has moved from the environment into the TOE.

2.5.2 SPD Statement

The security problem definition (SPD) of the current ST in chapter 4 contains the security problem definition of the claimd PP [JCSPP] without any additional items.

2.5.3 Security Objectives Statement

The security objectives statement for the TOE in the current ST includes all the security objectives for the TOE of the PP [JCSPP], but with the following changes:

The following objectives from the operational environment: OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.SCP.IC have been moved into the TOE. They are identified by O.SCP.RECOVERY, O.SCP.SUPPORT and O.SCP.IC. The content remains unchanged.

A new objective for the operational environment has been introduced (OE.SYM_KEY_GEN) to support symmetric key generation.

This satisfies the ‘demonstrable’ conformance claim of the [JCSPP] as overall objectives of the PP have not changed in the ST.

2.5.4 Security Requirements Statement

No additional SFRs have been introduced.

2.6 Conformance Statement

This ST has *demonstrable* conformance to the claimed PP [JCSPP].

3 Security Aspects

This chapter describes the main security issues of the Java Card System and its environment addressed in this Protection Profile, called “security aspects”, 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. For instance, we will define hereafter the following aspect:

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

TSFs must be continuously active in one way or another; this is called “OPERATE”. The Protection Profile may include an assumption, called “A.OPERATE”, stating that it is assumed that the TOE ensures continued correct operation of its security functions, and so on.

However, it may also include a threat, called “T.OPERATE”, to be interpreted as the negation of the statement #.OPERATE. In this example, this amounts to stating that an attacker may try to circumvent some specific TSF by temporarily shutting it down. The use of “OPERATE” is intended to ease the understanding of this document.

This section presents security aspects that will be used in the remainder of this document.

Some being quite general, we give further details, which are numbered for easier crossreference within the document. For instance, the two parts of #.OPERATE, when instantiated with an objective “O.OPERATE”, may be met by separate SFRs in the rationale. The numbering then adds further details on the relationship between the objective and those SFRs.

3.1 Confidentiality

3.1.1 #.CONFID-APPLI-DATA

Application data must be protected against unauthorised disclosure. This concerns logical attacks at runtime in order to gain read access to other application’s data.

3.1.2 #.CONFID-JCS-CODE

Java Card System code must be protected against unauthorised disclosure. Knowledge of the Java Card System code may allow bypassing the TSF. 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.

3.1.3 #.CONFID-JCS-DATA

Java Card System data must be protected against unauthorised 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 RE, the Java Card VM and the internal data of Java Card platform API classes as well.

3.2 Integrity

3.2.1 #.INTEG-APPLI-CODE

Application code must be protected against unauthorised modification. This concerns logical attacks at runtime in order to gain write access to the memory zone where executable code is stored. In post-issuance application loading, this threat also concerns the modification of application code in transit to the card.

3.2.2 #.INTEG-APPLI-DATA

Application data must be protected against unauthorised modification. This concerns logical attacks at runtime in order to gain unauthorised write access to application data. In 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.

3.2.3 #.INTEG-JCS-CODE

Java Card System code must be protected against unauthorised modification. This concerns logical attacks at runtime in order to gain write access to executable code.

3.2.4 #.INTEG-JCS-DATA

Java Card System data must be protected against unauthorised 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 RE, the Java Card VM and the internal data of Java Card API classes as well.

3.3 Unauthorised Executions

3.3.1 #.EXE-APPLI-CODE

Application (byte)code must be protected against unauthorised execution. This concerns (1) invoking a method outside the scope of the accessibility rules provided by the 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) unauthorised execution of a remote method from the CAD.

3.3.2 #.EXE-JCS-CODE

Java Card System bytecode must be protected against unauthorised execution. Java Card System bytecode includes any code of the Java Card RE or API. This concerns (1) invoking a method outside the scope of the accessibility rules provided by the 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.

3.3.3 #.FIREWALL

The Firewall shall ensure controlled sharing of class instances[6], and isolation of their data and code between packages (that is, controlled execution contexts) as well as between packages and the JCRE context. An applet shall not read, write, compare a piece of data belonging to an applet that is not in the same context, or execute one of the methods of an applet in another context without its authorization.

3.3.4 #.NATIVE

Because the execution of native code is outside of the JCS TSF scope, it must be secured so as to not provide ways to bypass the TSFs of the JCS.

Loading of native code, which is as well outside those TSFs, 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.

3.4 Bytecode Verification

3.4.1 #.VERIFICATION

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 files 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.

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

3.4.2 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 [JCVM22], [JVM], [JCBV]). The actual set of checks performed by the verifier is implementation-dependent, but shall at least enforce all the “must clauses” imposed in [JCVM22] on the bytecodes and the correctness of the CAP files’ format.

As most of the actual Java Card VMs 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. This ST assumes bytecode verification to be performed off card.

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 [JCVM22] §4.4), second, that the export files used to check and link the loaded applet have the corresponding correct counterpart on the card.

3.4.3 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. On-card bytecode verifier is out of the scope of this Protection Profile.

3.4.4 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 packages 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.

3.5 Card Management

3.5.1 #.CARD-MANAGEMENT

(1) The card manager (CM) shall control the access to card management functions such as the installation, update or deletion of applets. (2) The card manager shall implement the card issuer’s policy on the card.

3.5.2 #.INSTALL

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

3.5.3 #.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 Java Card RE, 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.x). A change of identity, especially standing for an administrative role (like an applet impersonating the Java Card RE), is a severe violation of the Security Functional Requirements (SFR). 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.

3.5.4 #.OBJ-DELETION

(1) Deallocation of objects 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.

3.5.5 #.DELETION

(1) Deletion of installed applets (or packages) 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 applets. 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 SFRs. 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 SFRs.

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 packages can be declared to be undeletable (for instance, the Java Card API packages), or the dependency between installed packages may forbid the deletion (like a package using super classes or super interfaces declared in another package).

3.6 Services

3.6.1 #.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 Java Card VM, or any other security-related event occurring during the execution of a TSF.

3.6.2 #.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.

3.6.3 #.RESOURCES

The TOE controls the availability of resources for the applications and enforces quotas and limitations in order to prevent unauthorised denial of service or malfunction of the TSFs. This concerns both execution (dynamic memory allocation) and installation (static memory allocation) of applications and packages.

3.6.4 #.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.

3.6.5 #.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.

3.6.6 #.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.

3.6.7 #.SCP

The smart card platform must be secure with respect to the SFRs. Then: (1) After a power loss, RF signal 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 SFRs to be bypassed or altered and does not allow access to other low-level functions than those made available by the packages of the Java Card 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. Finally it is required, that (7) the IC is designed in accordance with a well-defined set of policies and standards (for instance, those specified in [PP0035]), 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.

3.6.8 #.TRANSACTION

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

4 Security Problem Definition

4.1 Assets

Assets are security-relevant elements to be directly protected by the TOE. Confidentiality of assets is always intended with respect to un-trusted people or software, as various parties are involved during the first stages of the smart card product life-cycle; details are given in threats hereafter.

Assets may overlap, in the sense that distinct assets may refer (partially or wholly) to the same piece of information or data. For example, a piece of software may be either a piece of source code (one asset) or a piece of compiled code (another asset), and may exist in various formats at different stages of its development (digital supports, printed paper). This separation is motivated by the fact that a threat may concern one form at one stage, but be meaningless for another form at another stage.

The assets 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 weigh on it.

4.1.1 User Data

4.1.1.1 D.APP_CODE

The code of the applets and libraries loaded on the card.

To be protected from unauthorised modification.

4.1.1.2 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 unauthorised disclosure.

4.1.1.3 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 unauthorised modification.

4.1.1.4 D.PIN

Any end-user’s PIN.

To be protected from unauthorised disclosure and modification.

4.1.1.5 D.APP_KEYs

Cryptographic keys owned by the applets.

To be protected from unauthorised disclosure and modification.

4.1.2 TSF Data

4.1.2.1 D.JCS_CODE

The code of the Java Card System.

To be protected from unauthorised disclosure and modification.

4.1.2.2 D.JCS_DATA

The internal runtime data areas necessary for the execution of the Java Card VM, 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 unauthorised disclosure or modification.

4.1.2.3 D.SEC_DATA

The runtime security data of the Java Card RE, like, for instance, the AIDs used to identify the installed applets, the currently selected applet, the current context of execution and the owner of each object.

To be protected from unauthorised disclosure and modification.

4.1.2.4 D.API_DATA

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

To be protected from unauthorised disclosure and modification.

4.1.2.5 D.CRYPTO

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

To be protected from unauthorised disclosure and modification.

4.2 Threats

This section introduces the threats to the assets against which specific protection within the TOE or its environment is required. Several groups of threats are distinguished according to the configuration chosen for the TOE and the means used in the attack. The classification is also inspired by the components of the TOE that are supposed to counter each threat.

4.2.1 Confidentiality

4.2.1.1 T.CONFID-JCS-CODE

The attacker executes an application to disclose the Java Card System code. See #.CONFID-JCS-CODE for details.

Directly threatened asset(s): D.JCS_CODE.

4.2.1.2 T.CONFID-APPLI-DATA

The attacker executes an application to disclose data belonging to another application. See #.CONFID-APPLI-DATA for details.

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

4.2.1.3 T.CONFID-JCS-DATA

The attacker executes an application to disclose data belonging to the Java Card System. See #.CONFID-JCS-DATA for details.

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

4.2.2 Integrity

4.2.2.1 T.INTEG-APPLI-CODE

The attacker executes an application to alter (part of) its own or another application's code. See #.INTEG-APPLI-CODE for details.

Directly threatened asset(s): D.APP_CODE

4.2.2.2 T.INTEG-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

4.2.2.3 T.INTEG-APPLI-DATA

The attacker executes an application to alter (part of) another application's data. See #.INTEG-APPLI-DATA for details.

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

4.2.2.4 T.INTEG-JCS-DATA

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

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

4.2.2.5 T.INTEG-APPLI-CODE.LOAD

The attacker modifies (part of) its own or another application code when an application package is transmitted to the card for installation. See #.INTEG-APPLI-CODE for details.

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

4.2.2.6 T.INTEG-APPLI-DATA.LOAD

The attacker modifies (part of) the initialization data contained in an application package when the package is transmitted to the card for installation. See #.INTEG-APPLI-DATA for details.

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

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.

4.2.3 IdentityUsurpation

4.2.3.1 T.SID.1

An applet impersonates another application, or even the Java Card RE, in order to gain illegal access to some resources of the card or with respect to the end user or the terminal. See #.SID 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 and D.APP_KEYs.

4.2.3.2 T.SID.2

The attacker modifies the TOE's attribution of a privileged role (e.g. default applet and currently selected applet), which allows illegal impersonation of this role. See #.SID 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).

4.2.4 Unauthorised Execution

4.2.4.1 T.EXE-CODE.1

An applet performs an unauthorised execution of a method. See #.EXE-JCS-CODE and #.EXE-APPLI-CODE for details.

Directly threatened asset(s): D.APP_CODE.

4.2.4.2 T.EXE-CODE.2

An applet performs an execution of a method fragment or arbitrary data. #.EXE- JCS-CODE and #.EXE-APPLI-CODE for detail

Directly threatened asset(s): D.APP_CODE.

4.2.4.3 T.NATIVE

An applet executes a native method to bypass a TOE Security Function such as the firewall. See #.NATIVE for details.

Directly threatened asset(s): D.JCS_DATA.

4.2.4.4 T.EXE-CODE-REMOTE

The attacker performs an unauthorised remote execution of a method from the CAD. See #.EXE-APPLI-CODE for details.

Directly threatened asset(s): D.APP_CODE.

Note 2: This threat concerns version 2.2.x of the Java Card RMI, which allow external users (that is, other than on-card applets) to trigger the execution of code belonging to an on-card applet. On the contrary, T.EXE-CODE.1 is restricted to the applets under the TSF.

4.2.5 Denial of Service

4.2.5.1 T.RESSOURCES

An attacker prevents correct operation of the Java Card System through consumption of some resources of the card: RAM or NVRAM. See #.RESOURCES for details.

Directly threatened asset(s): D.JCS_DATA.

4.2.6 Card Management

4.2.6.1 T.INSTALL

The attacker fraudulently installs post-issuance of an applet on the card. This concerns either the installation of an unverified applet or an attempt to induce a malfunction in the TOE through the installation process. See #.INSTALL for details.

Directly threatened asset(s): D.SEC_DATA (any other asset may be jeopardized should this attack succeed, depending on the virulence of the installed application).

4.2.6.2 T.DELETION

The attacker deletes an applet or a package already in use on the card, or uses the deletion functions to pave the way for further attacks (putting the TOE in an insecure state). See #.DELETION for details.

Directly threatened asset(s): D.SEC_DATA and D.APP_CODE.

4.2.7 Services

4.2.7.1 T.OBJ-DELETION

The attacker keeps a reference to a garbage collected object in order to force the TOE to execute an unavailable method, to make it to crash, or to gain access to a memory containing data that is now being used by another application. See #.OBJ-DELETION for further details.

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

4.2.8 Miscellaneous

4.2.8.1 T.PHYSICAL

The attacker discloses or modifies the design of the TOE, its sensitive data or application code by physical (opposed to logical) tampering means. This threat includes IC failure analysis, electrical probing, unexpected tearing, and DPA. 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 the point (7) of the security aspect #.SCP, and all aspects related to confidentiality and integrity of code and data.

Note 3: In the rational of the PP the threat T.PHYSICAL was covered by the objective OE.SCP.IC of the environment. As the SCP has been included into the TOE in this Composite-Evaluation T.PHYSICAL is now covered by TOE objectives.

4.3 Organisational security policies

This section describes the organizational security policies to be enforced with respect to the TOE environment.

4.3.1 OSP.VERIFICATION

This policy shall ensure the consistency between the export files used in the verification and those used for installing the verified file. The policy must also ensure that no modification of the file is performed in between its verification and the signing by the verification authority. See #.VERIFICATION for details.

4.4 Assumptions

This section introduces the assumptions made on the environment of the TOE.

4.4.1 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.

4.4.2 A.DELETION

Deletion of applets through the card manager is secure. Refer to #.DELETION for details on this assumption.

.

4.4.3 A.APPLET

Applets loaded post-issuance do not contain native methods. The Java Card specification explicitly “does not include support for native methods” ([JCVM22], §3.3) outside the API.

5 Security Objectives

5.1 Security Objectives for the TOE

This section defines the security objectives to be achieved by the TOE.

5.1.1 Identification

5.1.1.1 O.SID

The TOE shall uniquely identify every subject (applet, or package) before granting it access to any service.

5.1.2 Execution

5.1.2.1 O.OPERATE

The TOE must ensure continued correct operation of its security functions. See #.OPERATE for details.

5.1.2.2 O.RESOURCES

The TOE shall control the availability of resources for the applications. See #.RESOURCES for details.

5.1.2.3 O.FIREWALL

The TOE shall ensure controlled sharing of data containers owned by applets of different packages, or the JCRE and between applets and the TSFs. See #.FIREWALL for details.

5.1.2.4 O.NATIVE

The only means that the Java Card VM 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 for details.

5.1.2.5 O.REALLOCATION

The TOE shall ensure that the re-allocation of a memory block for the runtime areas of the Java Card VM does not disclose any information that was previously stored in that block.

Note 4: 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 [JCVM222].

5.1.2.6 O.GLOBAL_ARRAYS_CONFID

The TOE shall ensure that the APDU buffer that is shared by all applications is always cleaned upon applet selection.

The TOE shall ensure that the global byte array used for the invocation of the install method of the selected applet is always cleaned after the return from the install method.

5.1.2.7 O.GLOBAL_ARRAYS_INTEG

The TOE shall ensure that only the currently selected application may have a write access to the APDU buffer and the global byte array used for the invocation of the install method of the selected applet.

5.1.3 Services

5.1.3.1 O.ALARM

The TOE shall provide appropriate feedback information upon detection of a potential security violation. See #.ALARM for details.

5.1.3.2 O.TRANSACTION

The TOE must provide a means to execute a set of operations atomically. See #.TRANSACTION for details.

5.1.3.3 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 for details.

5.1.3.4 O.PIN-MNGT

The TOE shall provide a means to securely manage PIN objects. See #.PIN_MNGT for details.

<u>Note 5:</u> 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.

5.1.3.5 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.

Note 6: O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION and O.CIPHER are actually provided to applets in the form of Java Card APIs. In addition to the Java Card API the Telecommunication Native Application provides APIs.

5.1.3.6 O.REMOTE

The TOE shall provide restricted remote access from the CAD to the services implemented by the applets on the card. This particularly concerns the Java Card RMI services introduced in version 2.2x of the Java Card platform.

5.1.4 Object Deletion

5.1.4.1 O.OBJ-DELETION

The TOE shall ensure the object deletion shall not break references to objects. See #.OBJ-DELETION for further details.

5.1.5 Applet Management

5.1.5.1 O.INSTALL

The TOE shall ensure that the installation of an applet performs as expected (See #.INSTALL for details).

5.1.5.2 O.LOAD

The TOE shall ensure that the loading of a package into the card is safe.

Note 7: 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.

5.1.5.3 O.DELETION

The TOE shall ensure that both applet and package deletion perform as expected. See #.DELETION for details.

5.1.6 Smart Card Platform

The SCP in this ST is part of the TOE. This is different from [JCSPP]. Security objectives for the Operational environment have been moved to objectives for the TOE.

5.1.6.1 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.

This security objective refers to the security aspect #.SCP(1): The smart card platform must be secure with respect to the SFRs. Then 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.

5.1.6.2 O.SCP.SUPPORT

This security objective refers to the security aspects 2,3,4 and 5 of #.SCP:

(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 lowlevel control accesses (segmentation fault detection).

5.1.6.3 O.SCP.IC

The SCP shall possess IC security features against physical attacks.

This security objective refers to the point (7) of the security aspect #.SCP:

- It is required that 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 8: O.SCP.IC covers also leackage attacks like DPA. The IC provides support against leackage attacks, which will be used by the TOE.

5.2 Security Objectives for the Operational Environment

This section introduces the security objectives to be achieved by the environment.

5.2.1 OE.CARD-MANAGEMENT

The card manager shall control the access to card management functions such as the installation, update or deletion of applets. It shall also implement the card issuer’s policy on the card.

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 TOE, which in turn shall very likely rely on the card manager for the effective enforcing of some of its security functions. Typically the card manager shall be in charge of the life cycle of the whole card, as well as that of the installed applications (applets).

The card manager should prevent that card content management (loading, installation, deletion) is carried out, for instance, at invalid states of the card or by non-authorised actors. It shall also enforce security policies established by the card issuer.

5.2.2 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 for details.

5.2.3 OE.APPLET

No applet loaded post-issuance shall contain native methods.

5.2.4 OE.SYM_KEY_GEN

Symmetric AES and 3-DES keys will be generated with a security level appropriate for its usage.

5.3 Security Objectives Rationale

The reader is refered to the Security Objective Rational of the [JCSPP] chapter 6.3. But as the objectives for the SCP (OE.SCP.RECOVERY, OE.SCP.SUPPORT and OE.SCP.IC) have been moved from the environment to the TOE itself some changes occur that will be described here:

- OE.SCP.RECOVERY must be replaced by O.SCP.RECOVERY
- OE.SCP.SUPPORT must be replaced by O.SCP.SUPPORT
- OE.SCP.IC must be replaced by O.SCP.IC

In chapter 6.3.1.8 *MISCELLANEOUS* of the [JCSPP] the sentence must be changed:

[JCSPP]	This ST
T.PHYSICAL Covered by OE.SCP.IC. Physical protections	**T.PHYSICAL** Covered by O.SCP.IC. Physical protections

rely on the underlying platform and are therefore an environmental issue.	rely on the underlying platform.

Compared to the [JCSPP] the objective for the environment OE.SYM_KEY_GEN has been added.

The threats T.CONFID-APPLI-DATA and T.INTEG-APPLI-DATA will be mapped to OE.SYM_KEY_GEN.

New mapping for 6.3.4 in [JCSPP]:

Threats	Security Objectives	Rational
T.CONFID-APPLI-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARDMANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.ALARM, O.TRANSACTION, O.CIPHER, O.PIN-MNGT, O.KEYMNGT, O.REALLOCATION, **OE.SYM_KEY_GEN**.	In the rational of 6.1.3 of [JCSPP] the OE.SYM_KEY_GEN will support the O.KEY-MNGT. In this ST rational it is also true that "keys … are particular cases of an applications's sensitive data …, that ask for appropriate management." Which will result in the objective O.KEY-MNGT etc. but in addition in OE.SYM_KEY_GEN.
T.INTEG-APPLI-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARDMANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.GLOBAL_ARRAYS_INTEG, O.ALARM, O.TRANSACTION, O.CIPHER, O.PIN-MNGT, O.KEYMNGT, O.REALLOCATION, **OE.SYM_KEY_GEN.**	See rational of T.CONFID-APPLI-DATA.

Security Objectives	Threats
OE.SYM_KEY_GEN	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA.

6 Security Requirements

6.1 Security Functional Requirements

This section states the security functional requirements for the Java Card System – Open configuration. For readability and for compatibility with the original Java Card System Protection Profile Collection – Standard 2.2 Configuration, requirements are arranged into groups. All the groups defined in the table below apply to this ST.

Group	Description
Core with Logical Channel (CoreG_LC)	The CoreG_LC contains the basic requirements concerning the runtime environment of the Java Card System implementing logical channels. This includes the firewall policy and the requirements related to the Java Card API. Logical channels are a Java Card specification version 2.2 feature. This group is the union of requirements from the Core (CoreG) and the Logical channels (LCG) groups defined in [PP/0305]. (cf. Java Card System Protection Profile Collection [JCSPPCol]).
Installation (InstG)	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.
Applet deletion (ADELG)	The ADELG contains the security requirements for erasing installed applets from the card, a feature introduced in Java Card specification version 2.2. It can also be used as a basis for any other application deletion requirements.
Remote Method Invocation (RMI)	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 specification version 2.2.
Object deletion (ODELG)	The ODELG contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card specification version 2.2 feature.
Secure carrier (CarG)	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 that do not support on-card static or dynamic bytecode verification, the installation of a package that has not been bytecode verified, or that has been modified after bytecode verification.

Subjects 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 or the PCD) and software components (like the application packages 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.

Subjects (prefixed with an “S”) are described in the following table:

Subject/Object/Information	Description
S.PACKAGE	A package is a namespace within the Java programming language that may contain classes and interfaces, and in the context of Java Card technology, it defines either a user library, or one or several applets.
S.JCRE	The runtime environment under which Java programs in a smart card are executed.
S.BCV	The bytecode verifier (BCV), which acts on behalf of the verification authority who is in charge of the bytecode verification of the packages. This subject is involved in the PACKAGE LOADING security policy defined in 6.1.6.
S.INSTALLER	The installer is the on-card entity which acts on behalf of the card issuer. This subject is involved in the loading of packages and installation of applets.
S.CARDMANAGER	The Card Manager charges Installer and Applet Deletion Manager to perform card content management operations (content loading, installation and deletion).
S.ADEL	The applet deletion manager which also acts on behalf of the card issuer. It may be an applet ([JCRE222], §11), but its role asks anyway for a specific treatment from the security viewpoint. This subject is unique and is involved in the ADEL security policy defined in 6.1.3.
S.CAD	The CAD represents the actor that requests, by issuing commands to the card, for RMI services. It could play the role of the off-card entity that communicates with the S.INSTALLER.
S.JCVM	The bytecode interpreter that enforces the firewall at runtime.
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.
S.APPLET	Any applet instance.
S.SPY	Any subject that potentially observe security critical operations to disclose keys and PINs.

Objects (prefixed with an “O”) are described in the following table:

O.JAVAOBJECT	Java class instance or array. It should be noticed that KEYS, PIN, arrays and applet instances are specific objects in the Java programming language.
O.CODE_PKG	The code of a package, including all linking information. On the Java Card platform, a package is the installation unit.
O.APPLET	Any installed applet, its code and data.
O.REMOTE_OBJ	A remote object is an instance of a class that implements one (or more) remote interfaces. A remote interface is one that extends, directly or indirectly, the interface java.rmi.Remote ([JCAPI222]).
O.ROR	A remote object reference. It provides information concerning: (i) the identification of a remote object and (ii) the Implementation class of the object or the interfaces implemented

	by the class of the object. This is the object's information to which the CAD can access.
O.REMOTE_MTHD	A method of a remote interface.
O.RMI_SERVICE	These are instances of the class javacardx.rmi.RMIService. They are the objects that actually process the RMI services.

Information (prefixed with an "I") is described in the following table:

I.DATA	JCVM Reference Data: objectref addresses of APDU buffer, JCRE-owned instances of APDU class and byte array for install method.
I.APDU	Any APDU sent to or from the card through the communication channel.
I.RORD	Remote object reference descriptors which provide information concerning: (i) the identification of the remote object and (ii) the implementation class of the object or the interfaces implemented by the class of the object. The descriptor is the only object's information to which the CAD can access.

Security attributes linked to these subjects, objects and information are described in the following table with their values:

Security attribute	**Description/Value**
Context	Package AID, or "Java Card RE"
Sharing	Standards, SIO, Java Card RE entry point, or global array
LifeTime	CLEAR_ON_DESELECT or PERSISTENT (*).
Selected Applet Context	Package AID, or "None"
Currently Active Context	Package AID, or "Java Card RE"
Package AID	The AID of each package indicated in the export file
Applet's version number	The version number of an applet (package) indicated in the export file
Dependent package AID	Allows the retrieval of the Package AID and Applet's version number ([JCVM222], §4.5.2)
Registred Applet	The AID of the applet instance registered on the card
Applet Selection Status	"Selected" or "Deselected"
LC Selection Status	Multiselectable, Non-multiselectable or "None" (logical channel)
ResidentPackages	Journals the list of AIDs of the packages already loaded on the card
ActiveApplets	The set of the active applets' AIDs. An active applet is an applet that is selected on at least one of the logical channels.
Remote	An object is said to be a Remote if it is an instance of a class that directly or indirectly implements the interface java.rmi.Remote
Owner	The Owner of an object is either the applet instance that created the object or the package (library) where it has been defined (these latter objects can only be arrays that initialize static fields

	of the package). The owner of a remote object is the applet instance that created the object.
ExportedInfo	Bollean (indicates whether the remote object is exportable or not).
Returned References	Lists the remote object references that have been sent to the CAD during the applet selection session. This attribute is implementation dependent.
Static References	Static fields of a package may contain references to objects. The Static References attribute records those references.
Class	Identifies the implementation class of the remote object.
Identifier	The Identifier of a remote object or method is a number that uniquely identifies the remote object or method, respectively.

(*) 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.

Operations (prefixed with "OP") are described in the following table. Each operation has 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(O.JAVAOBJECT, field)	Read/Write an array component.
OP.INSTANCE_FIELD(O.JAVAOBJECT, field)	Read/Write a field of an instance of a class in the Java programming language.
OP.INVK_VIRTUAL(O.JAVAOBJECT, method, arg1,...)	Invoke a virtual method (either on a class instance or an array object).
OP.INVK_INTERFACE(O.JAVAOBJECT, method, arg1,...)	Invoke an interface method.
OP.THROW(O.JAVAOBJECT)	Throwing of an object (athrow, see [JCRE222],§6.2.8.7)
OP.TYPE_ACCESS(O.JAVAOBJECT, class)	Invoke checkcast or instanceof on an object in order to access to classes(standard or shareable interfaces objects).
OP.JAVA(...)	Any access in the sense of [JCRE222], §6.2.8. It stands for one of the operations OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW, OP.TYPE_ACCESS.
OP.CREATE(Sharing, LifeTime) (*)	Creation of an object (new or makeTransient call).
OP.PUT(S1,S2,I)	Transfer a piece of information I from S1 to S2.
OP.DELETE_APPLET(O.APPLET,...)	Delete an installed applet and its objects, either logically or physically.
OP.DELETE_PCKG(O.CODE_PKG,...)	Delete a package, either logically or physically.
OP.DELETE_PCKG_APPLET(O.CODE_PKG,...)	Delete a package and its installed applets, either logically or physically.
OP.GET_ROR(O.APPLET,...)	Retrieves the initial remote object reference of a

	RMI based applet. This reference is the seed which the CAD client application needs to begin remote method invocations.
OP.INVOKE(O.RMI_SERVICE,...)	Requests a remote method invocation on the remote object.
OP.RET_RORD(S.JCRE,S.CAD,I.RORD)	Send a remote object reference descriptor to the CAD.

(*) For this operation, there is no accessed object. This rule enforces that shareable transient objects are not allowed. For instance, during the creation of an object, the JavaCardClass attribute's value is chosen by the creator.

6.1.1 CoreG LC Security Functional Requirements

This group is focused on the main security policy of the Java Card System, known as the firewall.

6.1.1.1 Firewall Policy

Except for the requirements explicitly introduced in what follows, this policy includes unchanged the functional requirements specified in the **FIREWALL *access control SFP*** of the group CoreG.

6.1.1.1.1 FDP_ACC.2/FIREWALL: Complete access control

6.1.1.1.1.1 *FDP_ACC.2.1/ FIREWALL*

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

Refinement:

The operations involved in this policy are:

- OP.CREATE
- OP.INVK_INTERFACE
- OP.INVK_VIRTUAL
- OP.JAVA
- OP.THROW
- OP.TYPE_ACCESS.

6.1.1.1.1.2 *FDP_ACC.2.2/FIREWALL*

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

Note 9:

It should be noticed that accessing array's components of a static array, and more generally fields and methods of static objects, is an access to the corresponding O.JAVAOBJECT.

6.1.1.1.2 FDP_ACF.1/FIREWALL Security attribute based access control

6.1.1.1.2.1 *FDP_ACF.1.1/FIREWALL*

The TSF shall enforce the **FIREWALL access control SFP** to objects based on the following:

Subject/Object	Security attributes
S.PACKAGE	**LC Selection Status**
S.JCRE	**Selected Applet Context**
S.JCVM	**Active Applets, Currently Active Context**
O.JAVAOBJECT	**Sharing, Context, LifeTime**

Table 6-1 Subjects and object of Firewall access control SFP

6.1.1.1.2.2 *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:

R.JAVA.1 ([JCRE222]]§6.2.8) An S.PACKAGE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW or OP.TYPE_ACCESS upon any O.JAVAOBJECT whose Sharing attribute has value "JCRE entry point" or "global array".

R.JAVA.2 ([JCRE222] §6.2.8) An S.PACKAGE may freely perform OP.ARRAY_ACCESS,OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE or OP.THROW upon any O.JAVAOBJECT whose Sharing attribute has value "Standard" and whose Lifetime attribute has value "PERSISTENT" only if O.JAVAOBJECT's Context attribute has the same value as the active context.

R.JAVA.3 ([JCRE222] §6.2.8.10) An S.PACKAGE may perform OP.TYPE_ACCESS upon an O.JAVAOBJECT whose Sharing attribute has value "SIO" only if O.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 ([JCRE222], §6.2.8.6,) An S.PACKAGE may perform OP.INVK_INTERFACE upon an O.JAVAOBJECT whose Sharing attribute has the value “SIO”, and whose Context attribute has the value “Package AID”, only if the invoked interface method extends the Sharable interface and one of the following conditions applies:

a) The value of the attribute Selection Status of the package whose AID is “Package AID” is “Multiselectable”,

b) The value of the attribute Selection Status of the package whose AID is “Package AID” is “Non-multiselectable”, and either “Package AID” is the value of the currently selected applet or otherwise “Package AID” does not occur in the attribute ActiveApplets,

R.JAVA.5 An S.PACKAGE may perform an OP.CREATE only if the value of the Sharing parameter is “Standard”.

6.1.1.1.2.3 *FDP_ACF.1.3/FIREWALL*

The TSF shall explicitly authorise access of subjects to objects based on the following additional rules:

1) The subject S.JCRE can freely perform OP.JAVA and OP.CREATE, with the exception given in FDP_ACF.1.4/FIREWALL, provided it is the Currently Active Context.

2) The only means that the subject S.JCVM shall provide for an application to execute native code is the invocation of a Java Card API method (through OP.INVK_INTERFACE or OP.INVK_VIRTUAL).

6.1.1.1.2.4 *FDP_ACF.1.4/FIREWALL*

The TSF shall explicitly deny access of subjects to objects based on the following additional rules:

1) Any subject with OP.JAVA upon an O.JAVAOBJECT whose LifeTime attribute has value “CLEAR_ON_DESELECT” if O.JAVAOBJECT’s Context attribute is not the same as the Selected Applet Context.

2) Any subject attempting to create an object by the means of OP.CREATE and a “CLEAR_ON_DESELECT” LifeTime parameter if the active context is not the same as the Selected Applet Context.

Note 10:

FDP_ACF.1.4/FIREWALL:

In the case of an array type, 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,
- 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 ([JCRE222], §6.1.3). 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).

([JCRE222], Glossary) Selected Applet Context. The Java Card RE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command with this applet's AID, the Java Card RE makes this applet the Selected Applet Context. The Java Card RE sends all APDU commands to the Selected Applet Context.

While the expression "Selected Applet Context" refers to a specific installed applet, the relevant aspect to the policy is the context (package AID) of the selected applet. In this policy, the "Selected Applet Context" is the AID of the selected package.

([JCRE222] §6.1.2.1) At any point in time, there is only one active context within the Java Card VM (this is called the Currently Active Context).

It should be noticed 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 to in this case. .

An applet instance is the currently selected applet instance only if it is processing the current command. There can only be one currently selected applet instance at a given time. ([JCRE22], §4).

6.1.1.1.3 FDP_IFC.1/JCVM Subset information flow control

6.1.1.1.3.1 FDP_IFC.1.1/JCVM

The TSF shall enforce the **JCVM information flow control SFP** on **S.JCVM, S.LOCAL, S.MEMBER, I.DATA and OP.PUT(S1, S2, I).**

Note 11:

It should be noticed that references of temporary Java Card RE entry points, which cannot be stored in class variables, instance variables or array components, are transferred from the internal memory of the Java Card RE (TSF data) to some stack through specific APIs (Java Card RE owned exceptions) or Java Card RE invoked methods (such as the process(APDU apdu)); these are causes of OP.PUT(S1,S2,I) operations as well.

6.1.1.1.4 FDP_IFF.1/JCVM Simple security attributes

6.1.1.1.4.1 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	Security attributes
S.JCVM	**Currently Active Context**

6.1.1.1.4.2 FDP_IFF.1.2/JCVM

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

- **An operation OP.PUT(S1, S.MEMBER, I.DATA) is allowed if and only if the Currently Active Context is "JCRE Card RE";**
- **other OP.PUT operations are allowed regardless of the Currently Active Context's value.**

6.1.1.1.4.3 FDP_IFF.1.3/JCVM

The TSF shall enforce **the additional information flow control SFP rules: none**

6.1.1.1.4.4 FDP_IFF.1.4/JCVM

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

6.1.1.1.4.5 FDP_IFF.1.5/JCVM

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

Note 12: The storage of temporary Java Card RE-owned objects references is runtime-enforced ([JCRE21], §6.2.8.1-3). Note that this policy essentially applies to the execution of bytecode.

6.1.1.1.5 FMT_MSA.1/JCVM Management of security attributes

6.1.1.1.5.1 FMT_MSA.1.1/JCVM

The TSF shall enforce the **FIREWALL access control SFP and the JCVM information flow control SFP** to restrict the ability to **modify** the security attributes **Currently Active Context and Active Applets** to **the Java Card VM (S.JCVM).**

Note 13: The modification of the Currently Active Context should be performed in accordance with the rules given in [JCRE222], §4 and [JCVM222], §3.4.

6.1.1.1.6 FDP_RIP.1/OBJECTS Subset residual information protection

6.1.1.1.6.1 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.**

Note 14: 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.

6.1.1.1.7 FMT_MSA.1/JCRE Management of security attributes

6.1.1.1.7.1 FMT_MSA.1.1/JCRE

The TSF shall enforce the **FIREWALL access control SFP** to restrict the ability to **modify** the security attributes: **Selected Applet Context** to **the Java Card RE.**

Note 15: The modification of the Selected Applet Context should be performed in accordance with the rules given in [JCRE21], §4 and [JCVM21], §3.4.

6.1.1.1.8 FMT_MSA.2/FIREWALL_JCVM Secure security attributes

6.1.1.1.8.1 FMT_MSA.2.1/FIREWALL_JCVM

The TSF shall ensure that only secure values are accepted for **all the security attributes of subjects and objects defined in the FIREWALL access control SFP and the JCVM information flow control SFP**.

6.1.1.1.9 FMT_MSA.3/FIREWALL Static attribute initialisation

6.1.1.1.9.1 FMT_MSA.3.1/FIREWALL

The TSF shall enforce the **FIREWALL access control SFP** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

6.1.1.1.9.2 FMT_MSA.3.2/FIREWALL

[Editorially refined]

The TSF shall not allow **any role** to specify alternative initial values to override the default values when an object or information is created.

Note 16:

FMT_MSA.3.1/FIREWALL

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 FIREWALL access control SFP permits the operation, 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 Currently Active Context, that is "Java Card RE".

- o 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 Java Card RE (and the Java Card virtual machine).

FMT_MSA.3.2/FIREWALL

The intent is that none of the identified roles has privileges with regard to the default values of the security attributes. It should be noticed that creation of objects is an operation controlled by the FIREWALL access control SFP. The operation shall fail anyway if the created object would have had security attributes whose value violates FMT_MSA.2.1/FIREWALL_JCVM.

6.1.1.1.10 FMT_MSA.3/JCVM Static attribute initialisation

6.1.1.1.10.1 FMT_MSA.3.1/JCVM

The TSF shall enforce the **JCVM information flow control SFP** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

6.1.1.1.10.2 FMT_MSA.3.2/JCVM

[Editorially refined]

The TSF shall not allow **any role** to specify alternative initial values to override the default values when an object or information is created.

6.1.1.1.11 FMT_SMR.1/JCRE Security roles

6.1.1.1.11.1 FMT_SMR.1.1/JCRE

The TSF shall maintain the roles:

- **the Java Card RE (JCRE)**
- **the Java Card VM (JCVM).**

6.1.1.1.11.2 FMT_SMR.1.2/JCRE

The TSF shall be able to associate users with roles.

6.1.1.1.12 FMT_SMF.1/JCRE Specification of Management Functions

6.1.1.1.12.1 FMT_SMF.1.1/JCRE

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

- **modify the Currently Active Context and the Selected Applet Context,**
- **modify the list of registered applets' AID.**

6.1.1.2 APPLICATION PROGRAMMING INTERFACE

The following SFRs are related to the Java Card API.

Not the whole set of cryptographic algorithms defined in [JCAPI222] is implemented. The implemented functions AES, DES and RSA are part of the TSF. The RNG is only indirectly being used. Other implemented cryptographic algorithms that are not described here are not part of the TSF.

6.1.1.2.1 FCS_CKM.1 Cryptographic key generation

6.1.1.2.1.1 *FCS_CKM.1.1*

The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm**:** **RSA-CRT and RSA-Key Generator** and specified cryptographic key sizes **1024 up to 2048 bit** that meet the following **list of standards: [JCAPI222], [AIS20] K3**.

Note 17:

The asymmetric keys can be generated and diversified in accordance with [JCAPI222] specification in classes KeyBuilder and KeyPair (at least Session key generation).

This component has been instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI222]).

Symmetric key generation for 3-DES and AES are supported by the operational environment (see OE.SYM_KEY_GEN). The further management of the symmetric keys is handeled in FCS_CKM.2, FCS_CKM.3 and FCS_CKM.4.

6.1.1.2.2 FCS_CKM.2 Cryptographic key distribution

6.1.1.2.2.1 *FCS_CKM.2.1*

The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method

DESKey.setKey()/all set-methods of class RSAPrivateCrtKey, RSAPrivateKey and RSAPublicKey/AESKey.setKey()

that meets the following **list of standards: [JCAPI222].**

Note 18:

This component has been instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI222]).

6.1.1.2.3 FCS_CKM.3 Cryptographic key access

6.1.1.2.3.1 *FCS_CKM.3.1*

The TSF shall perform **key access to the 3-DES/RSA/AES keys** in accordance with a specified cryptographic KEY access method

- **DESKey.getKey()/AESKey.getKey()**
- **All get-methods of class RSAPrivateCrtKey, RSAPrivateKey and RSAPublicKey**

that meets the following **list of standards: [JCAPI222].**

Note 19:

This component has been instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([JCAPI222]).

6.1.1.2.4 FCS_CKM.4 Cryptographic key destruction

6.1.1.2.4.1 FCS_CKM.4.1

The TSF shall destroy cryptographic keys in accordance with a specified cryptographic KEY destruction method ***Key*.clearKey() and overwriting the keys with zeros** that meets the following **list of standards: [JCAPI222].**

Note 20:

This component has been instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([JCAPI222]).

6.1.1.2.5 FCS_COP.1 Cryptographic operation

6.1.1.2.5.1 FCS_COP.1.1

6.1.1.2.5.1.1 FCS_COP.1.1.1

The TSF shall perform **signature generation** in accordance with a specified cryptographic algorithm **RSA-CRT** and cryptographic key sizes **1024 up to 2048 bit** that meet the following **RSA: [PKCS1], [RFC2409].**

6.1.1.2.5.1.2 FCS_COP.1.1.2

The TSF shall perform **signature generation** in accordance with a specified cryptographic algorithm **RSA** and cryptographic key sizes **1024 up to 2016 bit** that meet the following **RSA: [PKCS1], [RFC2409].**

Note 21:

The maximum key length of 2016 bit is caused by a physical limitation of the crypto-coprocessor and the specific implementation.

6.1.1.2.5.1.3 FCS_COP.1.1.3

The TSF shall perform **signature verification** in accordance with a specified cryptographic algorithm **RSA** and cryptographic key sizes **1024 up to 2048 bit** that meet the following **RSA: [PKCS1], [RFC2409].**

6.1.1.2.5.1.4 FCS_COP.1.1.4

The TSF shall perform **MAC generation and verification** in accordance with a specified cryptographic algorithm **DES CBC-MAC** and cryptographic key sizes **112, 168 bit** that meet the following **[ISO9797], [PKCS5], [JCAPI222].**

6.1.1.2.5.1.5 FCS_COP.1.1.5

The TSF shall perform **MAC generation and verification** in accordance with a specified cryptographic algorithm **AES CBC-MAC** and cryptographic key sizes **128, 192, 256 bit** that meet the following **[JCAPI222].**

6.1.1.2.5.1.6 FCS_COP.1.1.6

The TSF shall perform **encryption/decryption** in accordance with a specified cryptographic algorithm **3-DES in CBC/ ECB mode** and cryptographic key sizes **112, 168 bit** that meet the following **list of standards: DES: [ISO9797], [JCAPI222], [PKCS5].**

6.1.1.2.5.1.7 FCS_COP.1.1.7

The TSF shall perform **encryption/decryption** in accordance with a specified cryptographic algorithm **AES in CBC/ECB mode** and cryptographic key sizes **128, 192, 256 bit** that meet the following **list of standards: [JCAPI222].**

6.1.1.2.5.1.8 FCS_COP.1.1.8

The TSF shall perform **decryption** in accordance with a specified cryptographic algorithm **RSA** and cryptographic key sizes **1024 up to 2016 bit** that meet the following **list of standards: [ISO14888], [PKCS1], [JCAPI222].**

Note 22:

The maximum key length of 2016 bit is caused by a physical limitation of the crypto-coprocessor and the specific implementation.

6.1.1.2.5.1.9 FCS_COP.1.1.9

The TSF shall perform **decryption** in accordance with a specified cryptographic algorithm **RSA-CRT** and cryptographic key sizes **1024 up to 2048 bit** that meet the following **list of standards: [ISO14888], [PKCS1], [JCAPI222].**

6.1.1.2.5.1.10 FCS_COP.1.1.10

The TSF shall perform **encryption** in accordance with a specified cryptographic algorithm **RSA** and cryptographic key sizes **1024 up to 2048 bit** that meet the following **list of standards: [ISO14888], [PKCS1], [JCAPI222].**

6.1.1.2.6 FDP_RIP.1/APDU Subset residual information protection

6.1.1.2.6.1 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.**

> **Note 23:** 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.

6.1.1.2.7 FDP_RIP.1/bArray Subset residual information protection

6.1.1.2.7.1 FDP_RIP.1.1/bArray

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following object: **the bArray object.**

> **Note 24:** 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.

6.1.1.2.8 FDP_RIP.1/ABORT Subset residual information protection

6.1.1.2.8.1 FDP_RIP.1.1/ABORT

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **any reference to an object instance created during an aborted transaction.**

> **Note 25:** The events that provoke the de-allocation of a transient object are described in [JCRE21], §5.1.

6.1.1.2.9 FDP_RIP.1/KEYS Subset residual information protection

6.1.1.2.9.1 FDP_RIP.1.1/KEYS

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **the cryptographic buffer (D.CRYPTO).**

> **Note 26:** 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 [JCAPI21].

6.1.1.2.10 FDP_RIP.1/TRANSIENT Subset residual information protection

6.1.1.2.10.1 FDP_RIP.1.1/TRANSIENT

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **any transient object.**

Note 27: The events that provoke the de-allocation of any transient object are described in [JCRE22], §5.1.

• 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 applets that are active in different logical channels. Multiselectable applet instances within a same package must share the transient memory segment if they are concurrently active ([JCRE22], §4.2.

6.1.1.2.11 FDP_ROL.1/FIREWALL Basic rollback

6.1.1.2.11.1 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 the **operations OP.JAVA and OP.CREATE** on the **object O.JAVAOBJECTs.**

6.1.1.2.11.2 FDP_ROL.1.2/FIREWALL

The TSF shall permit operations to be rolled back within the **scope of a select(), deselect(), process(), install() or uninstall() call, notwithstanding the restrictions given in [JCRE222], §7.7, within the bounds of the Commit Capacity ([JCRE222], §7.8), and those described in [JCAPI222].**

Note 28: FDP_ROL.1.2/FIREWALL Transactions are a service offered by the APIs to applets. It is also used by some APIs to guarantee the atomicity of some operation. Some operations of the API are not conditionally updated, as documented in [JCAPI21] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).

It should be noticed that the rollback within the scope of the uninstall() method only applies to Java Card platform, version 2.2.1 compliant TOEs.

6.1.1.3 Card Security Management

6.1.1.3.1 FAU_ARP.1/JCS Security alarms

6.1.1.3.1.1 FAU_ARP.1.1/JCS

The TSF shall take **one of the following actions:**

- **throw an exception,**
- **lock the card session,**
- **reinitialize the Java Card System and its data,**
- **other actions: none**

upon detection of a potential security violation.

Refinement: ***The*** *"p*otential security violation" stands for one of the following events:

- *CAP file* inconsistency,
- typing error in the operands of a bytecode,
- *applet* life cycle inconsistency,
- card tearing (unexpected removal of the Card out of the CAD) and power failure,
- abort of a transaction in an unexpected context, (see `abortTransaction()`,[JCAPI222] and ([JCRE222], §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).

Note 29: The list above has been extended to provide all relevant potential security violation upon the TOE should react on. 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. Note: **For the TOE in this ST bytecode verification is performed off-card**.

6.1.1.3.2 FDP_SDI.2 Stored data integrity monitoring and action

6.1.1.3.2.1 *FDP_SDI.2.1*

The TSF shall monitor user data stored within the TSF for **integrity errors** on all objects, based on the following attributes**: checksum integrity of cryptographic keys, PIN values and their associated security attributes.**

6.1.1.3.2.2 *FDP_SDI.2.2*

Upon detection of a data integrity error, the TSF shall **bring the card into a secure state**.

Note 30: No such requirement is mandatory in the specification.It is also recommended by the PP to monitor integrity errors in the code of Java Card applets.

For integrity sensitive application, their data shall be monitored (D.APP_I_DATA): applications may need to protect information against unexpected modifications, and explicitly control whether a piece of information has been changed between two accesses. For example, maintaining the integrity of an electronic purse's balance is extremely important because this value represents real money. Its modification must be controlled, for illegal ones would denote an important failure of the payment system. The applet developer could either use supporting functions of the Java Card API (class Checksum) or implement its own mechanisms in the application, which both is out of scope of this evaluation.

6.1.1.3.3 FPT_TDC.1 Inter-TSF basic TSF data consistency

6.1.1.3.3.1 *FPT_TDC.1.1*

The TSF shall provide the capability to consistently interpret **the CAP files, the bytecode and its data arguments** when shared between the TSF and another trusted IT product.

6.1.1.3.3.2 FPT_TDC.1.2

The TSF shall use

- **the rules defined in [JCVM222] specification;**
- **the API tokens defined in the export files of reference implementation;**

When interpreting the TSF data from another trusted IT product.

Note 31:

FPT_TDC.1.1:

The TOE is developed consistently with the JCS and the SCP functions, including memory management, I/O functions and cryptographic functions. For the following APIs API token exist in the export file of the TOE: the JC API, the GP API the STK and further telecommunication related APIs.

6.1.1.3.4 FPT_FLS.1/JCS Failure with preservation of secure state

6.1.1.3.4.1 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.**

Note 32:

The Java Card RE Context is the Current context when the Java Card VM begins running after a card reset ([JCRE21], §6.2.3) or after a proximity card (PICC) activation sequence ([JCRE222]). Behavior of the TOE on power loss and reset is described in [JCRE21], §3.6 and §7.1. Behavior of the TOE on RF signal loss is described in [JCRE222], §3.6.1.

6.1.1.3.5 FPR_UNO.1 Unobservability

6.1.1.3.5.1 FPR_UNO.1.1

The TSF shall ensure that **S.SPY** is unable to observe the operation **cryptographic operations / comparison operations** on **key values / PIN values** by **S.JCRE, S.Applet**.

Note 33: The corresponding application note from the PP has been reflected in the SFR.

6.1.1.4 AID Management

6.1.1.4.1 FMT_MTD.1/JCRE Management of TSF data

6.1.1.4.1.1 FMT_MTD.1.1/JCRE

The TSF shall restrict the ability to **modify** *the* **list of registered applets' AID** to **the JCRE**.

6.1.1.4.2 FMT_MTD.3/JCRE Secure TSF data

6.1.1.4.2.1 *FMT_MTD.3.1/JCRE*

The TSF shall ensure that only secure values are accepted for **the registered applets' AIDs**.

6.1.1.4.3 FIA_ATD.1/AID User attribute definition

6.1.1.4.3.1 *FIA_ATD.1.1/AID*

The TSF shall maintain the following list of security attributes belonging to individual users:

- **Package AID,**
- **Applet's version number,**
- **Registered applet AID,**
- **Applet Selection Status ([JCVM222], §6.5).**

Note 34: "Individual users" stand for applets.

6.1.1.4.4 FIA_UID.2/AID User identification before any action

6.1.1.4.4.1 *FIA_UID.2.1/AID*

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

Note 35: By users here it must be understood the ones associated to the packages (or applets) that 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.

The role Java Card RE defined in FMT_SMR.1 is attached to an IT security function rather than to a "user" of the CC terminology. The Java Card RE does not "identify" itself to the TOE, but it is part of it.

6.1.1.4.5 FIA_USB.1/AID User-subject binding

6.1.1.4.5.1 *FIA_USB.1.1 / AID*

The TSF shall associate the following user security attributes with subjects acting on the behalf of that user: **Package AID.**

6.1.1.4.5.2 *FIA_USB.1.2 / AID*

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 FMT_MSA.2.1/FIREWALL_JCVM and FMT_MSA.3.1/FIREWALL and corresponding notes.**

6.1.1.4.5.3 *FIA_USB.1.3 / AID*

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.3.1/FIREWALL.**

Note 36: The user is the applet and the subject is the S.PACKAGE. The subject security attribute "Context" shall hold the user security attribute "package AID".

6.1.2 InstG Security Functional Requirements

This group consists the SFRs related to the installation of the *applet*s, which addresses security aspects outside the runtime. The installation of *applet*s is a critical phase, which lies partially out of the boundaries of the *firewall*, and therefore requires specific treatment. In this ST, loading a *package* or installing an *applet* modeled as importation of user data (that is, user application's data) with its security attributes (such as the parameters of the *applet* used in the firewall rules).

6.1.2.1.1 FDP_ITC.2/Installer Import of user data with security attributes

6.1.2.1.1.1 *FDP_ITC.2.1/Installer*

The TSF shall enforce the **PACKAGE LOADING information flow control SFP** when importing user data, controlled under the SFP, from outside of the TOE.

6.1.2.1.1.2 *FDP_ITC.2.2/Installer*

The TSF shall use the security attributes associated with the imported user data.

6.1.2.1.1.3 *FDP_ITC.2.3/Installer*

The TSF shall ensure that the protocol used provides for the unambiguous association between the security attributes and the user data received.

6.1.2.1.1.4 *FDP_ITC.2.4/Installer*

The TSF shall ensure that interpretation of the security attributes of the imported user data is as intended by the source of the user data.

6.1.2.1.1.5 *FDP_ITC.2.5/Installer*

The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TOE:

Package loading is allowed only if, for each dependent package, its AID attribute is equal to a resident package AID attribute, the major (minor) Version attribute associated to the dependant package is lesser than or equal to the major (minor) Version attribute associated to the resident package ([JCVM222]**,§4.5.2).**

6.1.2.1.2 FMT_SMR.1/Installer Security roles

6.1.2.1.2.1 *FMT_SMR.1.1/Installer*

The TSF shall maintain the roles: **the Installer.**

6.1.2.1.2.2 *FMT_SMR.1.2/Installer*

The TSF shall be able to associate users with roles.

6.1.2.1.3 FPT_FLS.1/Installer Failure with preservation of secure state

6.1.2.1.3.1 *FPT_FLS.1.1/Installer*

The TSF shall preserve a secure state when the following types of failures occur: **the installer fails to load/install a package/applet as described in [JCRE222], §11.1.4.**

__Note 37:__ FAU_ARP.1 describes feedback information in case of potential security violations.

6.1.2.1.4 FPT_RCV.3/Installer Automated recovery without undue loss

6.1.2.1.4.1 *FPT_RCV.3.1/Installer*

When automated recovery from **power loss** is not possible, the TSF shall enter a maintenance mode where the ability to return to a secure state is provided.

6.1.2.1.4.2 *FPT_RCV.3.2/Installer*

For **reset, insufficient flash memory, failure in cryptographic safeguarding, package references (versions) mismatching,** the TSF shall ensure the return of the TOE to a secure state using automated procedures.

6.1.2.1.4.3 *FPT_RCV.3.3/Installer*

The functions provided by the TSF to recover from failure or service discontinuity shall ensure that the secure initial state is restored without exceeding **0%** for loss of TSF data or objects under the control of the TSF.

6.1.2.1.4.4 *FPT_RCV.3.4/Installer*

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

6.1.3 ADELG Security Functional Requirements

This group combines the SFRs related to the deletion of applets and/or packages and enforcing the applet deletion manager (ADEL) policy on security aspects outside the runtime. Deletion is a critical phase and therefore requires specific treatment.

6.1.3.1 Applet Deletion Manager Policy

6.1.3.1.1 FDP_ACC.2/ADEL Complete access control

6.1.3.1.1.1 *FDP_ACC.2.1/ADEL*

The TSF shall enforce the **ADEL access control SFP** on **S.ADEL, S.JCRE, S.JCVM, O.JAVAOBJECT, O.APPLET *and* O.CODE_PKG** and all operations among subjects and objects covered by the SFP.

Refinement:

The operations involved in the policy are:

- OP.DELETE_APPLET,
- OP.DELETE_PCKG,
- OP.DELETE_PCKG_APPLET.

6.1.3.1.1.2 *FDP_ACC.2.2/ADEL*

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

6.1.3.1.2 FDP_ACF.1/ADEL Security attribute based access control

6.1.3.1.2.1 *FDP_ACF.1.1/ADEL*

The TSF shall enforce the **ADEL access control SFP** to objects based on the following:

Subject/Object	Attributes
S.JCVM	ActiveApplets
S.JCRE	Selected Applet Context, Registered Applets, Resident Packages
O.CODE_PKG	Package AID, **Dependent Package AID, Static References**
O.APPLET	**Applet Selection Status**
O.JAVAOBJECT	**Owner, Remote**

Table 6-2 Security attributes associated to the subjects/objects under control of the ADEL access control policy

6.1.3.1.2.2 *FDP_ACF.1.2/ADEL*

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

In the context of this policy, an object O is reachable if and only one of the following conditions hold:
(1) the owner of O is a registered applet instance A (O is reachable from A),
(2) a static field of a resident package P contains a reference to O (O is reachable from P),
(3) there exists a valid remote reference to O (O is remote reachable),
(4) there exists an object O' that is reachable according to either (1) or (2) or (3) above and O' contains a reference to O (the reachability status of O is that of O').

The following access control rules determine when an operation among controlled subjects and objects is allowed by the policy:

R.JAVA.14 ([JCRE222], §11.3.4.1, Applet Instance Deletion); S.ADEL may perform OP.DELETE_APPLET upon an O.APPLET only if,

(1) S.ADEL is currently selected,

(2) there is no instance in the context of O.APPLET that is active in any logical channel and

(3) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance distinct from O.APPLET, or O.JAVAOBJECT is reachable from a package P, or ([JCRE222], §8.5) O.JAVAOBJECT is remote reachable.

R.JAVA.15 ([JCRE222],§11.3.4.1, Multiple Applet Instance Deletion) ;

S.ADEL may perform OP.DELETE_APPLET upon several O.APPLET only if,

(1) S.ADEL is currently selected,

(2) there is no instance of any of the O.APPLET being deleted that is active in any logical channel and

(2) every O.APPLET being deleted is deselected and

(3) there is no O.JAVAOBJECT owned by any of the O.APPLET being deleted such that either O.JAVAOBJECT is reachable from an applet instance distinct from any of those O.APPLET, or O.JAVAOBJECT is reachable from a package P, or ([JCRE222], §8.5) O.JAVAOBJECT is remote reachable.

R.JAVA.16 ([JCRE222], §11.3.4.2, Applet/Library Package Deletion);

S.ADEL may perform OP.DELETE_PCKG upon an O.CODE_PKG only if,

(1) S.ADEL is currently selected,

(2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PKGthat is an instance of a class that belongs to O.CODE_PKG, exists on the card and

(3) there is no resident package on the card that depends on O.CODE_PKG.

R.JAVA.17 ([JCRE222], §11.3.4.3, Applet Package and Contained Instances Deletion); S.ADEL may perform OP.DELETE_PCKG_APPLET upon an O.CODE_PKG only if,

(1) S.ADEL is currently selected,

(2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PKG, which is an instance of a class that belongs to

O.CODE_PKG exists on the card,

(3) there is no package loaded on the card that depends on O.CODE_PKG, and

(4) for every O.APPLET of those being deleted it holds that:

-(i) there is no instance in the context of O.APPLET that is active in any logical channel and

- (ii) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance not being deleted, or O.JAVAOBJECT is reachable from a package not being deleted, or ([JCRE222],§8.5)O.JAVAOBJECT is remote reachable.

6.1.3.1.2.3 *FDP_ACF.1.3/ADEL*

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

6.1.3.1.2.4 *FDP_ACF.1.4/ADEL*

[Editorially Refined] The TSF shall explicitly deny access of **any subject but `S.ADEL` to `O.CODE_PKG` or `O.APPLET` for the purpose of deleting them from the card.**

Note 38:

FDP_ACF.1.2/ADEL:

- This policy introduces the notion of reachability, which provides a general means to describe objects that are referenced from a certain applet instance or package.
- S.ADEL calls the "uninstall" method of the applet instance to be deleted, if implemented by the applet, to inform it of the deletion request. The order in which these calls and the dependencies checks are performed are out of the scope of this security target.

6.1.3.1.3 FMT_MSA.1/ADEL Management of security attributes

6.1.3.1.3.1 *FMT_MSA.1.1/ADEL*

The TSF shall enforce the **ADEL access control SFP** to restrict the ability to **modify** the security attributes **Registered Applets and Resident Packages** to **the Java Card RE.**

6.1.3.1.4 FMT_MSA.3/ADEL Static attribute initialization

6.1.3.1.4.1 *FMT_MSA.3.1/ADEL*

The TSF shall enforce the **ADEL access control SFP** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

6.1.3.1.4.2 *FMT_MSA.3.2/ADEL*

The TSF shall allow the **following role(s): none**, to specify alternative initial values to override the default values when an object or information is created**.**

6.1.3.1.5 FMT_SMR.1/ADEL Security roles

6.1.3.1.5.1 *FMT_SMR.1.1/ADEL*

The TSF shall maintain the roles: **the applet deletion manager.**

6.1.3.1.5.2 *FMT_SMR.1.2/ADEL*

The TSF shall be able to associate users with roles.

6.1.3.1.6 FMT_SMF.1/ADEL Specification of Management Functions

6.1.3.1.6.1 *FMT_SMF.1.1/ADEL*

The TSF shall be capable of performing the following management functions: **modify the list of registered applets' AIDs and the Resident Packages.**

6.1.3.2 Additional Deletion Requirements

6.1.3.2.1 FDP_RIP.1/ADEL Subset residual information protection

6.1.3.2.1.1 *FDP_RIP.1.1/ADEL*

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **applet instances and/or packages when one of the deletion operations in FDP_ACC.2.1/ADEL is performed on them.**

Note 39: 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 [JCRE222], §11.3.4.1, §11.3.4.2 and §11.3.4.3.

6.1.3.2.2 FPT_FLS.1/ADEL Failure with preservation of secure state

6.1.3.2.2.1 *FPT_FLS.1.1/ADEL*

The TSF shall preserve a secure state when the following types of failures occur: **the applet deletion manager fails to delete a package/applet as described in [JCRE222], §11.3.4.**

Note 40:

- The TOE may provide additional feedback information to the card manager in case of a potential security violation (see FAU_ARP.1).
- The Package/applet instance deletion must be atomic. The "secure state" refered to in the requirement must comply with the Java Card specification ([JCRE222],, §11.3.4.

6.1.4 RMIG Security Functional Requirements

This group specifies the policies that control access to the remote objects and the flow of information that takes place when the RMI service is used. The rules relate mainly to the lifetime of the remote references. Information concerning remote object references can be sent out of the card only if the corresponding remote object has been designated as exportable. Array parameters of remote method invocations must be allocated on the card as global arrays. Therefore, the storage of references to those arrays must be restricted as well. The JCRMI policy embodies both an access control and an information control policy.

6.1.4.1.1 FDP_ACC.2/JCRMI Complete access control

6.1.4.1.1.1 *FDP_ACC.2.1/JCRMI*

The TSF shall enforce the **JCRMI access control SFP** on **S.CAD, S.JCRE, O.APPLET, O.REMOTE_OBJ, O.REMOTE_MTHD, O.ROR, O.RMI_SERVICE** andall operations among subjects and objects covered by the SFP.

Refinement:

The operations involved in this policy are:

- OP.GET_ROR
- OP.INVOKE.

6.1.4.1.1.2 *FDP_ACC.2.2/JCRMI*

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

6.1.4.1.2 FDP_ACF.1/JCRMI Security attribute based access control

6.1.4.1.2.1 *FDP_ACF.1.1/JCRMI*

The TSF shall enforce the **JCRMI access control SFP** to objects based on the following:

Subject / Object	Attributes
S.JCRE	Selected Applet Context
O.REMOTE_OBJ	Owner, Class, Identifier, ExportedInfo
O.REMOTE_MTHD	Identifier
O.RMI_SERVICE	Owner, Returned References

Table 6-3 Security attributes associated to the objects of the JCRMI access control policy

6.1.4.1.2.2 *FDP_ACF.1.2/JCRMI*

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

R.JAVA.18: S.CAD may perform OP.GET_ROR upon O.APPLET only if O.APPLET is the currently selected applet, and there exists an O.RMI_SERVICE with a registered initial reference to an O.REMOTE_OBJ that is owned by O.APPLET.

R.JAVA.19: S.JCRE may perform OP.INVOKE upon O.RMI_SERVICE, O.ROR and O.REMOTE_MTHD, only if, O.ROR is valid (as defined in [JCRE222], §8.5) and belongs to the returned references of O.RMI_SERVICE, and the attribute Identifier of O.REMOTE_MTHD matches one of the remote methods in the class, indicated by the security attribute class, of the O.REMOTE_OBJ to which O.ROR makes reference.

6.1.4.1.2.3 *FDP_ACF.1.3/JCRMI*

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

6.1.4.1.2.4 *FDP_ACF.1.4/JCRMI*

[Editorially Refined] The TSF shall explicitly deny access of **any subject but S.JCRE to O.REMOTE_OBJ and O.REMOTE_MTHD for the purpose of performing a remote method invocation.**

Note 41:

FDP_ACF.1.2/JCRMI:

- The validity of a remote object reference is specified as a lifetime characterization. The security attributes involved in the rules for determining valid remote object references are the Returned References of the O.RMI_SERVICE and the ActiveApplets (see FMT_REV.1.1/JCRMI and FMT_REV.1.2/JCRMI). The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([JCRE222], §8.5.2 and [JCAPI222]).

6.1.4.1.3 FDP_IFC.1/JCRMI Subset information flow control

6.1.4.1.3.1 *FDP_IFC.1.1/JCRMI*

The TSF shall enforce the **JCRMI information flow control SFP** on **S.JCRE, S.CAD, I.RORD and OP.RET_RORD(S.JCRE,S.CAD,I.RORD)**.

Note 42:

FDP_IFC.1.1/JCRMI:

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.

A remote object reference descriptor is sent from the card to the CAD either as the result of a successful applet selection command ([JCRE222], §8.4.1), and in this case it describes, if any, the initial remote object reference of the selected applet; or as the result of a remote method invocation ([JCRE222],§8.3.5.1).

6.1.4.1.4 FDP_IFF.1/JCRMI Simple security attributes

6.1.4.1.4.1 *FDP_IFF.1.1/JCRMI*

The TSF shall enforce the **JCRMI information flow control SFP** based on the following types of subject and information security attributes:

Subject/Information	Security attributes
`I.RORD`	**ExportedInfo**

6.1.4.1.4.2 *FDP_IFF.1.2/JCRMI*

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

OP.RET_RORD(S.JCRE, S.CAD, I.RORD is permitted only if the attribute ExportedInfo I.RORD has the value "true" ([JCRE222], §8.5).

6.1.4.1.4.3 *FDP_IFF.1.3/JCRMI*

The TSF shall enforce the **additional information flow control SFP rules: none**.

6.1.4.1.4.4 *FDP_IFF.1.4/JCRMI*

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

6.1.4.1.4.5 *FDP_IFF.1.5/JCRMI*

The TSF shall explicitly deny an information flow based on the following rules: **An operation *OP.RET_RORD(S.JCRE, S.CAD, I.RORD)* is denied if the attribute ExportedInfo *I.RORD* has the value "false" ([JCRE222], §8.5).**

Note 43: The ExportedInfo of I.RORD indicates whether the O.REMOTE_OBJ which I.RORD identifies is exported or not (as indicated by the security attribute ExportedInfo of the O.REMOTE_OBJ).

6.1.4.1.5 FMT_MSA.1/EXPORT Management of security attributes

6.1.4.1.5.1 *FMT_MSA.1.1/EXPORT*

The TSF shall enforce the **JCRMI access control SFP** to restrict the ability to **modify** the security attributes: **ExportedInfo of O.REMOTE_OBJ** to **its owner applet.**

Note 44: 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 modifaction itself can be performed by the Java Card RE.

6.1.4.1.6 FMT_MSA.1/REM_REFS Management of security attributes

6.1.4.1.6.1 FMT_MSA.1.1/REM_REFS

The TSF shall enforce the **JCRMI access control SFP** to restrict the ability to **modify** the security attributes **Returned References of O.RMI_SERVICE** to **its owner applet.**

6.1.4.1.7 FMT_MSA.3/JCRMI Static attribute initialization

6.1.4.1.7.1 FMT_MSA.3.1/JCRMI

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

6.1.4.1.7.2 FMT_MSA.3.2/JCRMI

The TSF shall allow the **following role(s) none,** to specify alternative initial values to override the default values when an object or information is created.

Note 45:

FMT_MSA.3.1/JCRMI:

Remote 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.

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 "Java Card RE".

FMT_MSA.3.2/JCRMI:

The intent is to have none of the identified roles to have privileges with regards to the default values of the security attributes. It should be noticed that creation of objects is an operation controlled by the FIREWALL access control SFP.

6.1.4.1.8 FMT_REV.1/JCRMI Revocation

6.1.4.1.8.1 FMT_REV.1.1/JCRMI

[Editorially Refined] The TSF shall restrict the ability to revoke **the Returned References of O.RMI_SERVICE** to **the Java Card RE.**

6.1.4.1.8.2 FMT_REV.1.2/JCRMI

The TSF shall enforce the rules **that determine the lifetime of remote object references.**

Note 46: The rules are described in [JCRE222], §8.5

6.1.4.1.9 FMT_SMR.1/JCRMI Security roles

6.1.4.1.9.1 FMT_SMR.1.1/JCRMI

The TSF shall maintain the roles: **applet**.

6.1.4.1.9.2 *FMT_SMR.1.2/JCRMI*

The TSF shall be able to associate users with roles.

Note 47: Applets own Remote interface objects and may choose to allow or forbid their exportation, which is managed through a security attribute.

6.1.4.1.10 FMT_SMF.1/JCRMI Specification of Management Functions

6.1.4.1.10.1 *FMT_SMF.1.1/JCRMI*

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

- **modify the security attribute ExportedInfo of O.REMOTE_OBJ.**
- **modify the security attribute Returned References of O.RMI_SERVICE.**

6.1.5 ODELG Security Functional Requirements

The following requirements concern the object deletion mechanism. This mechanism is triggered by the applet that owns the deleted objects by invoking a specific API method.

6.1.5.1.1 FDP_RIP.1/ODEL Subset residual information protection

6.1.5.1.1.1 *FDP_RIP.1.1/ODEL*

The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **the objects owned by the context of an applet instance which triggered the execution of the method `javacard.framework.JCSystem.requestObjectDeletion()`.**

Note 48: Freed data resources resulting from the invocation of the method javacard.framework.JCSystem.requestObjectDeletion() may be reused. Requirements on deallocation after the invocation of the method are described in [JCAPI222].

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 mus be performed in between APDU command processing, and therefore no transaction can be in progress.

6.1.5.1.2 FPT_FLS.1/ODEL Failure with preservation of secure state

6.1.5.1.2.1 *FPT_FLS.1.1/ODEL*

The TSF shall preserve a secure state when the following type of failures occur: **the object deletion functions fail to delete all the unreferenced objects owned by the applet that requested the execution of the method**

Note 49: The TOE may provide additional feedback information to the card manager in case of potential security violation (see FAU_ARP.1).

6.1.6 CarG Security Functional Requirements

This group includes requirements for preventing the installation of a packages that has not been bytecode verified, or that has been modified after bytecode verification.

6.1.6.1.1 FCO_NRO.2/CM Enforced proof of origin

6.1.6.1.1.1 FCO_NRO.2.1/CM

The TSF shall enforce the generation of evidence of origin for transmitted **application packages** at all times.

6.1.6.1.1.2 FCO_NRO.2.2/CM

[Editorially Refined] The TSF shall be able to relate the **identity** of the originator of the information, and the **application package contained in** the information to which the evidence applies.

6.1.6.1.1.3 FCO_NRO.2.3/CM

The TSF shall provide a capability to verify the evidence of origin of information to **recipient** given **at the time when a package is received.**

6.1.6.1.2 FIA_UID.1/CM Timing of identification

6.1.6.1.2.1 FIA_UID.1.1/CM

The TSF shall allow **the sending of the APDU commands to initiate communication through the trusted channel** on behalf of the user to be performed before the user is identified.

6.1.6.1.2.2 FIA_UID.1.2/CM

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

6.1.6.1.3 FDP_IFC.2/CM Complete information flow control

6.1.6.1.3.1 FDP_IFC.2.1/CM

The TSF shall enforce the PACKAGE LOADING information flow control SFP on **S.INSTALLER, S.BCV, S.CAD and I.APDU** and all operations that cause that information to flow to and from subjects covered by the SFP.

6.1.6.1.3.2 FDP_IFC.2.2/CM

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

Note 50:

- The subjects covered by this policy are those involved in the loading of an application package by the card through a potentially unsafe communication channel.
- The operations that make information to flow between the subjects are those enabling to send a message through and to receive a message from the communication channel linking the card to the outside world. It is assumed that any message sent through the channel as clear text can be read by an attacker. Moreover, an attacker may capture any message sent through the communication channel and send its own messages to the other subjects.
- The information controlled by the policy is the APDUs exchanged by the subjects through the communication channel linking the card and the CAD. Each of those messages contain part of an application package that is required to be loaded on the card, as well as any control information used by the subjects in the communication protocol.

6.1.6.1.4 FDP_IFF.1/CM Simple security attributes

6.1.6.1.4.1 *FDP_IFF.1.1/CM*

The TSF shall enforce the **PACKAGE LOADING information flow control SFP** based on the following types of subject and information security attributes:

(1) The keys used by the subjects S.INSTALLER and S.CARDMANAGER acting on behalf of the card issuer to decrypt and verify received messages;

(2) Authentication retry counter.

6.1.6.1.4.2 *FDP_IFF.1.2/CM*

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

(1) The subject S.INSTALLER shall accept a message only if it comes from the subject S.CAD;

(2) The subject S.INSTALLER shall accept an application package only if it has received all the APDUs sent by the subject S.CAD without modification and in the right order.

6.1.6.1.4.3 *FDP_IFF.1.3/CM*

The TSF shall enforce the **additional information flow control SFP rules: none.**

6.1.6.1.4.4 *FDP_IFF.1.4/CM*

The TSF shall explicitly authorise an information flow based on the following rules:

The information flow is authorised according the relevant rules in Appendix E [GP22].

6.1.6.1.4.5 *FDP_IFF.1.5/CM*

The TSF shall explicitly deny an information flow based on the following rules:

The Information flow is denied if the authentication retry counter limit is exceeded.

6.1.6.1.5 FDP_UIT.1/CM Data exchange integrity

6.1.6.1.5.1 FDP_UIT.1.1/CM

The TSF shall enforce the **PACKAGE LOADING information flow control SFP** to **receive** user data in a manner protected from **modification, replay, insertion and deletion** errors.

6.1.6.1.5.2 FDP_UIT.1.2/CM

[Editorially Refined] The TSF shall be able to determine on receipt of user data, whether **modification, deletion, insertion, replay of some of the pieces of the application sent by the CAD** has occurred.

6.1.6.1.6 FMT_MSA.1/CM Management of security attributes

6.1.6.1.6.1 FMT_MSA.1.1/CM

The TSF shall enforce the **PACKAGE LOADING information flow control SFP** to restrict the ability to **modify, delete, reset** the security attributes **the keys used by the subjects to encrypt/decrypt and sign their messages and the authentication retry counter** to **the S.CARDMANAGER acting on behalf of the card issuer.**

6.1.6.1.7 FMT_MSA.3/CM Static attribute initialization

6.1.6.1.7.1 FMT_MSA.3.1/CM

The TSF shall enforce the **PACKAGE LOADING information flow control SFP** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

6.1.6.1.7.2 FMT_MSA.3.2/CM

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

6.1.6.1.8 FMT_SMR.1/CM Security roles

6.1.6.1.8.1 FMT_SMR.1.1/CM

The TSF shall maintain the roles: **the installer, the card acceptance device.**

6.1.6.1.8.2 FMT_SMR.1.2/CM

The TSF shall be able to associate users with roles.

6.1.6.1.9 FMT_SMF.1/CM Specification of Management Functions

6.1.6.1.9.1 FMT_SMF.1.1/CM

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

<u>**Modification of the security attributes Card Life Cycle State and Security Level.**</u>

6.1.6.1.10 FTP_ITC.1/CM Inter-TSF trusted channel

6.1.6.1.10.1 *FTP_ITC.1.1/CM*

The TSF shall provide a communication channel between itself and another 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.

6.1.6.1.10.2 *FTP_ITC.1.2/CM*

[Editorially Refined] The TSF shall permit **the CAD placed in the card issuer secured environment** to initiate communication via the trusted channel.

6.1.6.1.10.3 *FTP_ITC.1.3/CM*

The TSF shall initiate communication via the trusted channel for **loading/installing a new application package on the card.**

6.2 Security assurance requirements

The security assurance requirement level is EAL4 augmented with AVA_VAN.5 and ALC_DVS.2.

6.3 Security Requirements Rationale

The reader is refered to the Security Requirements Rational of the [JCSPP] chapter 7.3 as far as possible. Especially the objectives for the SCP moved from the environment to the TOE causes some changes in the ST as described below.

6.3.1 Objectives

6.3.1.1 Security Objectives for the TOE

As in [JCSPP] chapter 7.3.1.1.1 to 7.3.1.1.5, but in addition with the following chapter:

6.3.1.1.1 Smart Card Platform Objectives

6.3.1.1.2 O.SCP.RECOVERY

This objective is met by FDP_ROL.1/FIREWALL and FAU_ARP.1 and supported by the cleanup functions: FDP_RIP.1.1/bArray, FDP_RIP.1.1/ADEL, FDP_RIP.1.1/ODEL, FDP_RIP.1.1/OBJECTS, FDP_RIP.1.1/APDU, FDP_RIP.1.1/ABORT, FDP_RIP.1.1/KEYS, FDP_RIP.1.1/TRANSIENT.

FDP_ROL.1/FIREWALL describes the rollback of critical Java security operations. FAU_ARP.1 guarantees that the TOE takes appropriate actions upon detection of a potential security violation, like card tearing, power failure and the abort of a transaction in an unexpected context. The cleanup functions support the protection of the secure state.

The extension of the TOE by the SCP together with ADV_ARC.1 allows a mapping of O.SCP.RECOVERY to these SFRs.

6.3.1.1.3 O.SCP.SUPPORT

This objective is met by FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1. As the SCP is part of the TOE and ADV_ARC.1 ensures that the TOE is implemented, so that the security features of the TSF cannot be bypassed and that it protects itself from tampering by untrusted active enteties ADV_ARC.1 requires that the SCP does not allow access to other not intended low-level functions and the SFRs require also SCP support for low-level cryptographic functions. For the same reason (ADV_ARC.1 and the extended TOE including the SCP) the objective of support for atomicity is met by FDP_ROL.1/FIREWALL. All SFRs will map to the objective for data storage in persistent technology memory.

6.3.1.1.4 O.SCP.IC

This objective is met by all SFRs. As the SCP is part of the TOE and ADV_ARC.1 ensures that the TOE is implemented, so that the security features of the TSF cannot be bypassed and that it protects itself from tampering by untrusted active enteties all SFRs meet the objective for a tamper resistant against commonly employed techniques.

6.3.2 Rational tables of Security Objectives and SFRs

Table 7 of chapter 7.3.2 in the [JCSPP] must be extended by the following rows

Security Objectives	Security Functional Requirements	Rationale

O.SCP.RECOVERY	FDP_RIP.1.1/bArray, FDP_RIP.1.1/ADEL, FDP_RIP.1.1/ODEL, FDP_RIP.1.1/OBJECTS, FDP_RIP.1.1/APDU, FDP_RIP.1.1/ABORT, FDP_RIP.1.1/KEYS, FDP_RIP.1.1/TRANSIENT, FDP_ROL.1/FIREWALL, FAU_ARP.1	Section 6.3.1
O.SCP.SUPPORT	FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FMT_MSA.1/JCVM, FDP_RIP.1/OBJECTS, FMT_MSA.1/JCRE, FMT_SMR.1, FMT_SMF.1, FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_ROL.1/FIREWALL, FAU_ARP.1/JCS, FDP_SDI.2, FPT_TDC.1, FPT_FLS.1/JCS, FPR_UNO.1, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FIA_ATD.1/AID, FIA_UID.2/AID, FIA_USB.1/AID, FDP_ITC.2/Installer, FMT_SMR.1/Installer, FPT_FLS.1/Installer, FPT_RCV.3/Installer, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL, FMT_SMF.1/ADEL, FDP_ACC.2/ADEL, FDP_ACF.1/ADEL, FDP_RIP.1/ADEL, FPT_FLS.1/ADEL, FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, , FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI, FMT_SMF.1/JCRMI, FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ODEL, FPT_FLS.1/ODEL, FMT_MSA.1/CM, FMT_MSA.3/CM, FMT_SMR.1/CM, FMT_SMF.1/CM, FCO_NRO.2/CM, FIA_UID.1/CM, FDP_IFC.2/CM, FDP_IFF.1/CM, FDP_UIT.1/CM, FTP_ITC.1/CM	Section 6.3.1
O.SCP.IC	FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FMT_MSA.1/JCVM, FDP_RIP.1/OBJECTS, FMT_MSA.1/JCRE, FMT_SMR.1, FMT_SMF.1, FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_ROL.1/FIREWALL, FAU_ARP.1/JCS, FDP_SDI.2, FPT_TDC.1, FPT_FLS.1/JCS, FPR_UNO.1, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FIA_ATD.1/AID, FIA_UID.2/AID, FIA_USB.1/AID, FDP_ITC.2/Installer, FMT_SMR.1/Installer, FPT_FLS.1/Installer, FPT_RCV.3/Installer, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL, FMT_SMF.1/ADEL, FDP_ACC.2/ADEL, FDP_ACF.1/ADEL,	Section 6.3.1

	FDP_RIP.1/ADEL, FPT_FLS.1/ADEL, FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI, FMT_SMF.1/JCRMI, FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ODEL, FPT_FLS.1/ODEL, FMT_MSA.1/CM, FMT_MSA.3/CM, FMT_SMR.1/CM, FMT_SMF.1/CM, FCO_NRO.2/CM, FIA_UID.1/CM, FDP_IFC.2/CM, FDP_IFF.1/CM, FDP_UIT.1/CM, FTP_ITC.1/CM	

Table 6-4 SCP objectives / SFR mapping

Table 8 of chapter 7.3.2 in the [JCSPP] must be extended by the additional SCP objectives. The updated table is Table 6-5 in this document.

Functional requirements for the TOE	Security objectives of the PP	Additional Security Objectives in the ST
FDP_ACC.2/FIREWALL	See Table 8 in the [JCSPP].	O.SCP.SUPPORT, O.SCP.IC
FDP_ACF.1/FIREWALL		
FDP_IFC.1/JCVM		
FDP_IFF.1/JCVM		
FMT_MSA.1/JCVM		
FMT_MSA.1/JCRE		
FMT_SMR.1/JCRE		
FMT_SMF.1/JCRE		
FCS_CKM.1		
FCS_CKM.2		
FCS_CKM.3		
FCS_CKM.4		
FCS_COP.1		
FDP_ROL.1/FIREWALL		O.SCP.RECOVERY, O.SCP.SUPPORT, O.SCP.IC
FDP_RIP.1/OBJECTS		
FDP_RIP.1/APDU		
FDP_RIP.1/bArray		
FDP_RIP.1/ABORT		
FDP_RIP.1/KEYS		
FDP_RIP.1/ADEL		
FDP_RIP.1/TRANSIENT		

FDP_RIP.1/ODEL		
FAU_ARP.1/JCS		
		O.SCP.SUPPORT, O.SCP.IC
FDP_SDI.2		
FPT_TDC.1		
FPT_FLS.1/JCS		
FPR_UNO.1		
FMT_MTD.1/JCRE		
FMT_MTD.3/JCRE		
FIA_ATD.1/AID		
FIA_UID.2/AID		
FIA_USB.1/AID		
FDP_ITC.2/Installer		
FMT_SMR.1/Installer		
FPT_FLS.1/Installer		
FPT_RCV.3/Installer		
FDP_ACC.2/ADEL		
FDP_ACF.1/ADEL		
FMT_MSA.1/ADEL		
FMT_MSA.3/ADEL		
FMT_SMR.1/ADEL		
FMT_SMF.1/ADEL		
FPT_FLS.1/ADEL		
FDP_ACC.2/JCRMI		
FDP_ACF.1/JCRMI		
FDP_IFC.1/JCRMI		
FDP_IFF.1/JCRMI		
FMT_MSA.1/EXPORT		
FMT_MSA.1/REM_REFS		
FMT_MSA.3/JCRMI		
FMT_REV.1/JCRMI		
FMT_SMR.1/JCRMI		
FMT_SMF.1/JCRMI		
FPT_FLS.1/ODEL		
FCO_NRO.2/CM		
FIA_UID.1/CM		
FDP_IFC.2/CM		
FDP_IFF.1/CM		
FDP_UIT.1/CM		
FMT_MSA.1/CM		
FMT_MSA.3/CM		
FMT_SMR.1/CM		
FMT_SMF.1/CM		
FTP_ITC.1/CM		

Table 6-5 SFRs and Security Objectives (extended version of Table 8 of the **[JCSPP]**).

6.3.3 Dependencies

See chapter 7.3.3 of [JCSPP].

The dependencies for symmetric key distribution (FCS_CKM.2, FCS_CKM.3, FCS_CKM.4 and FCS_COP.1) are met by the objective for the operational environment OE.SYM_KEY_GEN. For symmetric keys it is a recommendation for the user to generate und use appropriate 3-DES and AES keys.

6.3.4 Rational for the Security Assurance Requirements

See chapter 7.3.4 of [JCSPP].

6.3.5 AVA_VAN.5 Advanced methodical vulnerability analysis

See chapter 7.3.5 of [JCSPP].

6.3.6 ALC_DVS.2 Sufficiency of security measures

See chapter 7.3.6 of [JCSPP].

7 TOE summary specification

This chapter provides information about the mechanisms that the TOE uses to fulfil the SFRs.

7.1 TOE Mechanisms

7.1.1 M1: TRANSACTION

This mechanism ensures the rollback process[7]. It provides assurance in the Java objects update in flash memory.

1. The rollback operation restores the original values of the persistent objects (modified during the transaction) and clears the dedicated transaction area.
2. The TOE permits the rollback of the OP.JAVA, OP.CREATE on the O.JAVAOBJECTs.

7.1.2 M2: ACCESS_CONTROL

This mechanism provides control for the TOE. It is in charge of the FIREWALL access control SFP and the JCVM information flow control SFP.

1. The TOE enforces the Firewall access control SFP and the JCVM information flow policy to control the flow of information between subjects.
2. The TOE restricts the ability to modify the list of registered applets and packages AID to the JCRE and maintains 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.
3. The TOE requires each user to identify itself before allowing any other TSF-mediated actions on behalf of that user and associates the following user security attributes with subjects acting on behalf of that user: Package AID.
4. Only secure values are accepted for security attributes.
5. The ability to modify the Currently Active Context and the Active Applets is restricted to the Java Card VM (S.JCVM). The ability to modify the Selected Applet Context is restricted to the Java Card RE (S.JCRE).
6. The TOE provides Inter-TSF data consistency. The TOE uses rules stated in FPT_TDC.1.2 when interpreting the TSF data from another trusted IT product.

[7] Java Card technology supports a transaction mechanism with commit and rollback capability to guarantee that complex operations can be accomplished atomically; either they successfully complete or their partial results are not put into effect.

7.1.3 M3: CRYPTO

This mechanism controls all the operations related to the cryptographic key management and cryptographic operations.

This mechanism is composed of:

1. Key Generation for RSA-CRT and RSA according to [JCAPI222]; [AIS20] K3.
2. Key access and distribution: the TOE provides 3-DES key (112, 168 bit), RSA (1024 up to 2048 bit) and AES (128, 192, 256 bit) access (6.1.1.2.3.1) and distribution (6.1.1.2.2.1) in accordance with [JCAPI222].
3. Key destruction: the TOE provides a cryptographic 3-DES, RSA and AES key destruction method (6.1.1.2.4.1).
4. Encryption/decryption and sign/verify in accordance with a specified cryptographic algorithm 3-DES in CBC/ ECB mode, RSA, RSA-CRT and AES in CBC/ECB mode as specified in 6.1.1.2.5.1 FCS_COP.1.1.

7.1.4 M4: INTEGRITY

This mechanism provides a means to check the integrity of checksummed data stored in flash memory.

1. This mechanism initializes the checksum of cryptographic keys, PIN values and their associated security attributes.
2. The TOE monitors cryptographic keys, PIN values and their associated security attributes stored within the TSF for integrity errors by checksum testing.
3. Upon detection of a data integrity error on cryptographic keys, PIN values and their associated security attributes the TOE will throw an exception and prevent the usage of this key/PIN or switch to an endless loop. This is a secure state.

7.1.5 M5: SECURITY

This mechanism ensures a secure state of information, the non-observability of operations on it and the unavailability of previous information content upon deallocation/allocation.

1. The TOE throws an exception, locks the card session or reinitialises the Java Card System and its JCRE data upon detection of a potential security violation and preserves a secure state.
2. The TOE ensures that an attacker is unable to observe cryptographic operations / comparison operations on key values / PIN values.

3. The TOE ensures that any previous information content of a resource is made unavailable upon deallocation of the resource from the bArray object, any reference to an object instance created during an aborted transaction and the cryptographic buffer. At least upon allocation of the APDU buffer any previous information content is made unavailable.

7.1.6 M6: APPLET

This mechanism ensures the secure loading of a *package* or installing of an *applet by S.CAD and the secure* deletion of *applet*s and/or *packages by S.ADEL.*

1. The TOE enforces the PACKAGE LOADING information flow control SFP when importing user data by loading of a *package* or installing of an *applet* e.g. and maintains the installer role.
2. The TOE uses the security attributes associated with the loaded *packages* or installed *applets*.
3. The *package*, loading is allowed by the TOE only if, for each dependent *package*, its *AID* attribute is equal to a resident package *AID* attribute, the major (minor) Version attribute associated to the former is equal (less than or equal) to the major (minor) Version attribute associated to the latter ([JCVM222],§4.5.2).
4. When the installer fails to load/install a package/applet it preserves a secure state as described in [JCRE222] §10.1.4. and enters a maintenance mode where the ability to return the TOE to a secure state is provided for reset, insufficient flash memory, failure in cryptographic safeguarding, package references (versions) mismatching
5. The TOE enforces the ADEL access control SFP.
6. The TOE restricts the ability to modify the Registered Applets and Resident Packages to the JCRE.
7. The TOE ensures that any previous information content of a resource is made unavailable upon the deallocation of the resource from applet instances and/or packages and from the objects owned by the context of an applet instance which triggered the execution of the method `javacard.framework.JCSystem.requestObjectDeletion()` or if deletion operations according to ADEL access control SFP occur.
8. The TOE preserves a secure state when the applet deletion manager fails to delete a package/applet as described in [JCRE222], §11.3.4 and the object deletion functions fail to delete all the unreferenced objects owned by the applet that requested the execution of the method.

7.1.7 M7: RMI

This mechanism ensures secure remote method invocation features, which provides a new protocol of communication between the terminal and the applets.

1. The TOE enforces the JCRMI access control SFP to control the access to remote objects when the RMI service is used.
2. The TOE enforces the JCRMI information flow control SFP to control the flow of information that takes place when the RMI service is used.
3. The TOE enforces the JCRMI access control SFP and the JCVM information flow control SFP to restrict the ability to modify
   - the security attribute ExportedInfo of O.REMOTE_OBJ to its owner applet,
   - the security attribute Returned References of O.RMI_SERVICE to its owner applet

   and provides restrictive default values for security attributes that are used to enforce the SFP.
4. The TOE restricts the ability to revoke the Returned References security attribute of an O.RMI_SERVICE to the JCRE.
5. The TOE enforces the rules that determine the lifetime of remote object references.
6. The TOE maintains the role applet and associates users with this role.

7.1.8 M8: CARRIER

This mechanism ensures secure downloading of applications on the card.

1. The TOE enforces the generation of evidence of origin for transmitted application packages at all times.
2. The TOE is able to relate the identity of the originator of the information, and the application package contained in the information to which the evidence applies.
3. The TOE provides a capability to verify the evidence of origin of information to the recipient given at the time it is received.
4. The TOE allows the sending of the APDU commands to initiate communication through the trusted channel on behalf of the user to be performed before the user is identified.
5. The TOE requires each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.
6. The TOE enforces the PACKAGE LOADING information flow control SFP to secure the reception of an application package by the card through a potentially unsafe communication channel.

7. The TOE enforces the PACKAGE LOADING information flow control SFP to provide restrictive default values for security attributes that are used to enforce the SFP.
8. The TOE maintains the roles: S.INSTALLER, S.CAD and associates users with these roles.
9. The TOE provides a communication channel between itself and a remote 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.
10. The TOE permits the CAD placed in the card issuer secured environment to initiate communication through the trusted channel.
11. The TOE requires communication through the trusted channel for installing a new application package on the card.
12. The TOE is capable of modifying the security attributes Card Life Cycle State and Security Level.

7.2 Fulfilment of the SFRs

The following table shows the mapping of the SFRs to mechanisms of the TOE.

	M1: TRANSACTION	**M2: ACCESS_CONTROL**	**M3: CRYPTO**	**M4: INTEGRITY**	**M5: SECURITY**	**M6: APPLET**	**M7: RMI**	**M8: CARRIER**
FDP_ACC.2.1/ FIREWALL **FDP_ACC.2.2/FIREWALL**		1						
FDP_ACF.1.1/FIREWALL **FDP_ACF.1.2/ FIREWALL** **FDP_ACF.1.3/FIREWALL** **FDP_ACF.1.4/FIREWALL**		1						
FDP_IFC.1.1/JCVM		1						

	M1: TRANSACTION	M2: ACCESS_CONTROL	M3: CRYPTO	M4: INTEGRITY	M5: SECURITY	M6: APPLET	M7: RMI	M8: CARRIER
FDP_IFF.1.1/JCVM, FDP_IFF.1.2/JCVM, FDP_IFF.1.3/JCVM, FDP_IFF.1.4/JCVM FDP_IFF.1.5/JCVM,		1						
FMT_MSA.1.1/JCVM		5						
FDP_RIP.1.1/OBJECTS					3			
FMT_MSA.1.1/JCRE		5						
FMT_MSA.2.1/FIREWALL_JCVM		4						
FMT_MSA.3.1/FIREWALL FMT_MSA.3.2/FIREWALL		2						
FMT_MSA.3.1/JCVM FMT_MSA.3.1/JCVM							3	
FMT_SMR.1.1/JCRE, FMT_SMR.1.2/JCRE		1						
FMT_SMF.1.1/JCRE		2, 5						
FCS_CKM.1.1			1, 4					
FCS_CKM.2.1			2					
FCS_CKM.3.1			2					
FCS_CKM.4.1			3					
FCS_COP.1.1			1, 4					
FDP_RIP.1.1/APDU					3			
FDP_RIP.1.1/bArray					3			
FDP_RIP.1.1/ABORT					3			
FDP_RIP.1.1/KEYS					3			
FDP_RIP.1.1/TRANSIENT					3			
FDP_ROL.1.1/FIREWALL, FDP_ROL.1.2/FIREWALL	1, 2							

	M1: TRANSACTION	M2: ACCESS_CONTROL	M3: CRYPTO	M4: INTEGRITY	M5: SECURITY	M6: APPLET	M7: RMI	M8: CARRIER
FAU_ARP.1.1/JCS					1[8]			
FDP_SDI.2.1, FDP_SDI.2.2				1, 2, 3				
FPT_TDC.1.1, FPT_TDC.1.2		6						
FPT_FLS.1.1/JCS					1			
FPR_UNO.1.1					2			
FMT_MTD.1.1/JCRE		2						
FMT_MTD.3.1/JCRE		4						
FIA_ATD.1.1/AID		2						
FIA_UID.2.1/AID		3						
FIA_USB.1.1		2, 3						
FIA_USB.1.2		2, 4						
FIA_USB.1.3		5						
FDP_ITC.2.1/Installer, FDP_ITC.2.2/Installer, FDP_ITC.2.3/Installer, FDP_ITC.2.4/Installer, FDP_ITC.2.5/Installer						1 2 3		
FMT_SMR.1.1/Installer FMT_SMR.1.2/Installer						1		
FPT_FLS.1.1/Installer						4		
FPT_RCV.3.1/Installer, FPT_RCV.3.2/Installer, FPT_RCV.3.3/Installer, FPT_RCV.3.4/Installer,						4		
FDP_ACC.2.1/ADEL,						5		

[8] The numbers in the table give the corresponding component of the mechanism covering the requirement.

	M1: TRANSACTION	M2: ACCESS_CONTROL	M3: CRYPTO	M4: INTEGRITY	M5: SECURITY	M6: APPLET	M7: RMI	M8: CARRIER
FDP_ACC.2.2/ADEL								
FDP_ACF.1.1/ADEL, FDP_ACF.1.2/ADEL, FDP_ACF.1.3/ADEL, FDP_ACF.1.4/ADEL						5		
FMT_MSA.1.1/ADEL, FMT_MSA.3.1/ADEL, FMT_MSA.3.2/ADEL						6		
FMT_SMR.1.1/ADEL, FMT_SMR.1.2/ADEL						8		
FMT_SMF.1.1/ADEL						6		
FDP_RIP.1.1/ADEL						7		
FPT_FLS.1.1/ADEL						8		
FDP_ACC.2.1/JCRMI FDP_ACC.2.2/JCRMI							1	
FDP_ACF.1.1/JCRMI FDP_ACF.1.2/JCRMI FDP_ACF.1.3/JCRMI FDP_ACF.1.4/JCRMI							1	
FDP_IFC.1.1/JCRMI FDP_IFF.1.1/JCRMI FDP_IFF.1.2/JCRMI FDP_IFF.1.3/JCRMI FDP_IFF.1.4/JCRMI FDP_IFF.1.5/JCRMI							2	
FMT_MSA.1.1/EXPORT FMT_MSA.1.1/REM_REFS							3	
FMT_MSA.3.1/JCRMI FMT_MSA.3.2/JCRMI							3	
FMT_REV.1.1/JCRMI							4	
FMT_REV.1.2/JCRMI							5	

	M1: TRANSACTION	M2: ACCESS_CONTROL	M3: CRYPTO	M4: INTEGRITY	M5: SECURITY	M6: APPLET	M7: RMI	M8: CARRIER
FMT_SMR.1.1/JCRMI **FMT_SMR.1.2/JCRMI**							6	
FMT_SMF.1.1/JCRMI							3	
FDP_RIP.1.1/ODEL						7		
FPT_FLS.1.1/ODEL						8		
FCO_NRO.2.1/CM								1
FCO_NRO.2.2/CM								2
FCO_NRO.2.3/CM								3
FIA_UID.1.1/CM								4
FIA_UID.1.2/CM								5
FDP_IFC.2.1/CM **FDP_IFC.2.2/CM**								6
FDP_IFF.1.1/CM **FDP_IFF.1.2/CM** **FDP_IFF.1.3/CM** **FDP_IFF.1.4/CM** **FDP_IFF.1.5/CM**								6
FDP_UIT.1.1/CM **FDP_UIT.1.2/CM**								6
FMT_MSA.1.1/CM								6
FMT_MSA.3.1/CM **FMT_MSA.3.2/CM**								7
FMT_SMR.1.1/CM **FMT_SMR.1.2/CM**								8
FMT_SMF.1.1/CM								12
FTP_ITC.1.1/CM								9
FTP_ITC.1.2/CM								10
FTP_ITC.1.3/CM								11

Table 7-1 Mapping of SFRs to mechanisms of TOE

8 Annexe A: Definitions and acronyms

8.1 Definitions

This section provides definitions about terms frequently used in this document. The definition of the Common Criteria related terms is specified in [CC1], § 4.

AID	For this definitions please see [JCSPP].
APDU	
APDU buffer	
Applet	
Applet deletion manager	
BVC	
CAD	
CAP file	
Class	
Context	
Current context	
Currently selected applet	
Default applet	
DPA	
Embedded Software	
Firewall	
Installer	
Interface	
Java Card RE	
Java Card RE Entry Point	
Java Card RMI	
Java Card System	
Java Card VM	

Logical channel

NVRAM

Object deletion

Package

PCD

PICC

RAM

SCP

Sharable interface

SIO

Subject

SWP

Transient object

User

ITSEF	IT Security evaluation laboratory approved by the certification body of the country where the evaluation is performed. It manages Common Criteria evaluations of platforms and secure applications.
Application developer	Company which develops secure or standard applications that shall be loaded onto the platform.
Application provider	Entity or organization which manages the application and the associated services. It can be a bank, a transport operator or a third-party service provider.
Certification body	State office which manages the Common Criteria certification of the platforms and the secure applications evaluated by an ITSEF. In France, it is the ANSSI.
Issuer	See TOE issuer
Mobile operator	Owner of the (U)SIM Java Card Platform which also manages a mobile network. The platform ensures that it is the only one, after authentication, to be able to manage applications (loading, installation, and deletion).
Smart Card IC Provider	Company which manages the development and the production of the IC.
Smart Card manufacturer	Company which manages the manufacturing of the (U)SIM card, especially in the the of the IC Card manufacturing phase product.
Smart Card personalizer	Company which performs the personalization of the (U)SIM Java Card Platform.
TOE issuer	See mobile operator
Trusted third-party	Trusted entity or verification authority which signs application after validation or certification. The application signature is verified on the card at loading by an agent of the trusted third-party which is present on the platform.

Validation laboratory	Accredited Security laboratory approved by the Mobile Operator which manages the validation of the standard applications.
Verification authority	see Trusted third party

8.2 Acronyms

CC	Common Criteria
ADELG	Applet Deletion Group
APDU	Application Protocol Data Unit
API	Application Programming Interface
BCVG	Byte Code Verifier Group
CarG	Secure Carrier Group
CoreG	Core Group
CoreG LC	Core with Logical Channels Group
CEM	Common Evaluation Methodology
EAL	Evaluation Assurance Level
ETR_COMP	Rapport d'évaluation pour la Composition
IC	Integrated Circuit
ISCI	International Security Certification Initiative
ITSEF	Information Technology Security Evaluation Facility
JCS	Java Card System
LCG	Logical Channel Group
OS	Operating system
OSP	Organisational Security Policy
PP	Protection Profile
RMIG	Remote Method Invocation Group
SCP	Smart Card Platform
SF	Security Function
SIM	Subscriber Identity Module
SSCD	Secure Signature Creation Device
ST	Security Target
STK	SIM Toolkit
SWP	Single Wire Protocol
TOE	Target Of Evaluation
TSF	TOE Security Functions

USIM	Universal Subscriber Identity Module

9 Annexe B: References

[AES]	*Federal Information Processing Standards Publication 197, ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001*
[AIS20]	*Anwendungshinweise und Interpretationen zum Schema (AIS), AIS 20, Version 1, Stand 02.12.1999, Funktionalitätsklassen und Evaluierungsmethodologie für deterministische Zufallszahlengeneratoren, Zertifizierungsstelle des BSI.*
[CC1]	*Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model; CCMB-2006-09-001, Version 3.1, Revision 3, July 2009, CCMB-2009-07-001.*
[CC2]	*Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Components; CCMB-2007-09-002, Version 3.1, Revision 3, July 2009, CCMB-2009-07-002.*
[CC3]	*Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements; CCMB-2007-09-003, Version 3.1, Revision 3, July 2009, CCMB-2009-07-003*
[CCDB]	*Composite product evaluation for Smart Cards and similar devices, September 2007, Version 1.0, Revision 1, CCDB-2007-09-001*
[CCTrans]	*Guide for the Transition from Common Criteria Version 2.3 to Common Criteria Version 3.1 for ADV Requirements, Version 1.0, November 2007.*
[CEM]	*Common Methodology for Information Technology Security Evaluation, Evaluation Methodology; CCMB-2009-07-004, Version 3.1, Revision 3,July 2009*
[DES]	*National Institute of Standards and Technology, Data Encryption Standard, Federal Information Processing Standards Publication 46-3, October 25, 1999.*
[ETSI102221]	*UICC-Terminal interface; Physical and logical characteristics, V6.14.0 (2007-07)*
[GP22]	*GlobalPlatform Card Specification Version 2.2, May 2006*
[ISO7816-3]	*ISO/IEC 7816-3 (2006): Identification cards – Integrated circuit cards – Part 3: Cards with contacts – Electrical interface and transmission protocols.*
[ISO7816-4]	*ISO/IEC 7816-4 (2005): Identification cards – Part 4: Organization, security and commands for interchange*
[ISO7816-6]	*ISO/IEC 7816-6 (2003): Identification cards – integrated circuit cards – Part 6: Interindustry data elements for interchange.*
[ISO9797]	*ISO/IEC 9797-1:1999: Information technology – Security techniques –Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher.*
[ISO14888]	*Information technology – Security techniques – Digital signatures with appendix: ISO/IEC 14888.*
[JCVM222]	*Java Card 2.2.2 Virtual Machine (JCVM) Specification. October 2005. Published by Sun Microsystems, Inc.*
[JCAPI222]	*Java Card 2.2.2 Application Programming Interface. March 2006. Published by Sun Microsystems, Inc.*
[JCRE222]	*Java Card 2.2.2 Runtime Environment (JCRE) Specification. March 2006. Published by Sun Microsystems, Inc.*

[JCBV]	*Java Card 2.2 Off-Card Verifier. June 2002. White paper. Published by Sun Microsystems, Inc.*
[JCSPP]	*Java Card System Protection Profile Version 2.6, Open Configuration, ANSSI PP ANSSI-CC-PP-2010/03, Sun Microsystems Inc ., April 2010.*
[JCSPPCol]	*Java Card Protection Profile Collection, Version 1.0b, August 2003, registered and certified by the French certification body (ANSSI) underthe following references: [PP/0303] "Minimal Configuration", [PP/0304] "Standard 2.1.1 Configuration", [PP/0305] "Standard 2.2 Configuration" and [PP/0306] "Defensive Configuration".*
[JVM]	*The Java Virtual Machine Specification. Lindholm, Yellin. ISBN 0-201-43294-3.*
[PKCS1]	*PKCS #1: RSA Encryption Standard – An RSA Laboratories Technical Note, Version 1.5, Revised November 1, 1993*
[PKCS5]	*PKCS #5: Password-Based Encryption Standard, Version 1.5.*
[RFC2409]	*The Internet Key Exchange (IKE) document RFC 2409 (STD 1).*
[RSA]	*RFC 3447, J. Jonsson, B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", February 2003*
[STL]	*Security Target of S3FS91J/S3FS91H/S3FS91V/S3FS93I, 32-bits RISC Microcontroller For Smart Card with SWP, Version 1.0, 30th November 2009, Samsung Electronics.*
[SUPP]	*Supporting Document, Mandatory Technical Document, Composite product evaluation for Smart Cards and similar devices, September 2007, Version 1.0, CCDB-007-09-001.*

End of Document