Mokard Safe 2.2:
ASE-Security Target Lite

Incard Srl
81925 Marcianise (Ce) - Italy
Zona Industriale Marcianise Sud

Revision History:

Edition-Rev.	Subject	Issued by:	Authorized by:	Date
A-0	Issue			

EXECUTIVE SUMMARY

Java Card™ technology was tailored in order to enable programs written in the Java™ programming language to run on smart cards and other resource–constrained devices. Due to these constraints, every component of the original Java™ platform was significantly reduced. On the other hand, smart cards require specific security features beyond the scope of the standard Java platform. For instance, even the legitimate holder of a credit card should not be able to tamper with some of the data contained on the card (for instance, its credit value). Moreover, just like browsers are to distrust downloaded applets to protect the local resources, the Java Card™ environment must prevent the terminal or even the installed applets, which may come from various sources, from accessing vendor–specific confidential data.

A security evaluation, according to a standard such as the Common Criteria scheme, is an appropriate answer to meet this need for enhanced security. It provides assurance measures to gauge risks and induced costs, discover weak points prior their exploitation by hostile agents, and finally grants a level of certification according to recognized standards of industry for future reference. It also highlights numerous points that may easily be overlooked although they are extremely relevant to the security of a Java Card technology-based implementation.

This document presents a set of security requirements for the Java Card platform ("Java Card System") that comply with the security requirements related to Java Card 2.2 Standard Configuration as stated in the Java Card System Protection profile collection v1.0b. Java Card System Protection profile collection v1.0b define four protection profile registered at the French Common Criteria Certification Authority DCSSI. The PP addressed in this document is registered with the code PP/0305.

The emphasis is mainly laid on those issues related to the **firewall** mechanisms and **bytecode verification**, the two cornerstones of the security architecture for the Java Card platform ("Java Card security architecture"). The protection endorsed by the firewall to applications loaded in a multi-application platform as the one provided by Java Card technology ultimately relies on those applications having passed the checks performed by a bytecode verifier. Indeed, without bytecode verification, a Java Card technology-based application ("Java Card applications") may misbehave as any application written in native code. The mutual support between these components also depends on the contribution provided by other constituents of the product, such as the underlying platform or the application installer program. The clarification of the nature of these dependencies, which were implicit in the functional specification, is the key to achieve a **safe and coherent interaction** of the components, that is, to build security interoperability on top of functional interoperability. The already existing Protection Profiles (such as SCSUG's "S*mart Card PP*" and Eurosmart's "*Smart Card IC with Multi-Application Secure Platform*") for the underlying platform, as well as Global Platform's "*Card Security Requirements Specification*" on card management are also considered.

Finally, this document proposes some additional security features to identify and deal with security–sensitive data. That would extend specific protections that are applied to cryptographic keys or PIN code; for instance, the integrity of the balance in an e–purse application requires similar "strong" protection. These features should normalize the secure programming of applets containing sensitive data (such as banking applications).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 ASSOCIATED DOCUMENTS

1.1 REFERENCE DOCUMENTS

[CC1] *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model*. Version 2.1. August 1999. CCIMB-99-031.

[CC2] *Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements*. Version 2.1. August 1999. CCIMB-99-032.

[CC3] *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance requirements*. Version 2.1. August 1999. CCIMB-99-033.

[CEM] *Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology*. Version 1.0. August 1999. CEM-99/045.

[JCVM21] *Java Card 2.1.1 Virtual Machine (JCVM) Specification*. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCAPI21] *Java Card 2.1.1 Application Programming Interface*. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCRE21] *Java Card 2.1.1 Runtime Environment (JCRE) Specification*. Revision 1.0. May 18, 2000. Published by Sun Microsystems, Inc.

[JCVM22] *Java Card 2.2 Virtual Machine (JCVM) Specification*. June 2002. Published by Sun Microsystems, Inc.

[JCAPI22] *Java Card 2.2 Application Programming Interface*. June 2002. Published by Sun Microsystems, Inc.

[JCRE22] *Java Card 2.2 Runtime Environment (JCRE) Specification*. June 2002. Published by Sun Microsystems, Inc.

[JCBV] *Java Card 2.1.2 Off-Card Verifier*. January 2001. White paper. Published by Sun Microsystems, Inc.

[JAVASPEC] *The Java Language Specification*. Gosling, Joy and Steele. ISBN 0-201-63451-1.

[JVM] *The Java Virtual Machine Specification*. Lindholm, Yellin. ISBN 0-201-43294-3.

[JCSPPC] *Java Card System Protection Profile Collection Ver 1.0b August 2003*.

1.2 RELATED DOCUMENTS

The following list is in no way exhaustive.

[SCSUG-3] *Smart Card Protection Profile*. Smart Card Security User Group. Version 3.0, September 9, 2001. Registered and Certified by Bundesamt für Sicherheit in der Informationstechnik (BSI) under the reference BSI-PP-000 3-2001. Registered and Certified by the French Certification Body under the reference PP/0103. Registered and Certified by the Canadian Certification Body.

[PP9806] *Protection Profile Smart Card IC*. Version 2.0, Issue November 1998. Registered and Certified by the French Certification Body under the reference PP/9806.

[PP0010] *Protection Profile Smart Card IC with Multi-Application Secure Platform*. Version 2.0, Issue November 2000. Registered and Certified by the French Certification Body under the reference PP/0010.

[SSVG-1.0] *Smartcard IC Platform Protection Profile*. Version 1.0, July 2001. Registered and Certified by Bundesamt für Sicherheit in der Informationstechnik (BSI) under the reference BSI-PP-0002.

[GP] *GlobalPlatform Card Specification*, Version 2.1.1, March 2003.

[CSRS] *GlobalPlatform Card Security Requirements Specification*, Version 1.0, May 2003.

2 INTRODUCTION

This chapter identifies the document and the references it cites, presents its general structure, and introduces some key notions and notation conventions to be used in the following chapters.

2.1 ST IDENTIFICATION

2.1.1 Identification of the Document

ST Title:	Mokard Safe 2.2: ASE-Security Target
ST Version:	A-0
Date of Modification:	14 October 2005
TOE:	Mokard Safe 2.2
TOE Version:	2.4.0
Relative Product Name:	Mokard Safe 2.2 on S3CJ9QD
Product Version:	2.4.0
IT Security Scheme:	German Scheme
Evaluation Body:	TUV Informationstechnik GmbH evaluation body
Certification Body:	Bundesamt fur Sicherheit in der Informationstechnik (BSI)
Assurance Level:	EAL 4 augmented

This ST has been written according to Common Criteria Version 2.1 (ISO 15408)

2.1.2 SECURITY TARGET OVERVIEW

The aim of this document is to describe the Security Target of "Mokard Safe 2.2".

"Mokard Safe 2.2" is an ST Incard smartcard platform developed in compliance with the most recent version of international standards. "Mokard Safe 2.2" is compliant with the standard Java Card 2.2.1. The TOE is the ST Incard implementation of this technology.

"Mokard Safe 2.2" addresses both the 2G and 3G mobile market by means of high secure SIM and USIM applications.

"Mokard Safe 2.2" supports Telecom Operators in developing and deploying mobile services requiring the highest level of security such as for example M-Commerce applications.

Platform security is proofed and internationally recognized by the international standard Common Criteria.

This security target is conformant to the Java Card System Protection Profile Collection Version 1.0b [JCSPPC] that is the state of the art on security for Java Card platforms. Java Card System Protection Profile Collection Version 1.0b has been sponsored by SUN Microsystems and has been reviewed and approved by the major Smart Card manufacturer.

Due to the high level of security guaranteed by "Mokard Safe 2.2" its usage span from the mobile applications to the financial and personal identity applications.

2.2 CC CONFORMANCE CLAIM

This ST is conformant with the Common Criteria Version 2.1 (ISO 15408):

- Part 2 [CC2],
- Part 3 [CC3].

The minimum strength of function is SOF-High.

The assurance level is EAL 4 augmented on ADV_IMP.2 and AVA_VLA. 3.

This ST is conformant to PP Java Card System Protection Profile Collection V1.0b in the configuration Java Card System Standard 2.2 version 1.0b, August 2003, registered at the French Certification Body (DCSSI) under the number PP/0305.

2.3 TYPOGRAPHIC CONVENTIONS

– This typeface is used to highlight those words that appear in the glossary. Example: applet.

– **This typeface** is used to highlight asset names. Example: **D.APP_CODE**.

– ***THIS TYPEFACE*** is used for those words referring to entities within the TSC or operations of security policies (Common Criteria terminology). Example: ***S.APPLET***.

– **`This typeface`** is used for keywords of the Java™ programming language, variables, method or field names, and so on. Example: a **`public static`** field.

– *THIS TYPEFACE* is used for the name of threats, objectives and assumptions. Example: *O.TODO*.

Finally, the following format of paragraph is used to remind Common Criteria components:

CC_FUNC^{al}_REQ^{t} **The TSF shall ensure this and that.**

3 TOE DESCRIPTION

This part of the document shall describe the TOE as an aid to the understanding of its security requirements, and shall address the product type and the general IT features of the TOE.

3.1 TOE ABSTRACT

The TOE is the schematically reported in Figure 1. The red dashed line delimits its perimeter.

The TOE is composed of the following parts:

- Java Card Runtime Environment,
- Installer.

Java Card Runtime Environment (JCRE) is the central component of a Java Card System. It consists of the following pieces:

- Java Card Virtual Machine (JCVM),
- Java Card API and its associated native methods,
- Remote Method Invocation (RMI) facilities,
- Logical Channels
- Object deletion facilities.

The Java Card virtual machine is the interpreter of the Java Card bytecode. It enforces the separation between the applications by means of the applets firewall and enables secure data sharing.

The Java Card API and its associated native methods provides the Java Card developer of the methods and the services to interact with the platform resources.

RMI facilities are all the facilities on the card enabling a client application running on the CAD platform to invoke a method on a remote object on the card. RMI is a new functionality introduced by Java Card 2.2 and has been listed separately by the API in this description notwithstanding RMI facilities are available to applet developers by means of standard API.

Logical Channels is a new feature of Java Card 2.2 that enables the opening of up to four simultaneous session with the card, one per logical channel.

Object Deletion facilities are all the facilities provided by the Java Card 2.2 ensuring that any unreferenced object owned by the current context is deleted and the associated space is recovered for reuse prior to the next card reset. This facilities is an optional Java Card 2.2 facilities.

The TOE also include the installer responsible for the installation of applets on the card. The installer loads and links packages on the card to a suitable form for the JCVM to execute the code they contain. It is a subsystem of the card manager and it could be seen as the portion of the card manager belonging to the TOE. Moreover the installer is responsible for the deletion of applets and packages and its associated data on the card. The deletion is controlled by means of what is called deletion manager.

All the java features embedded in "Mokard 3G" are compliant with Java Card specification Version 2.2 [JCAPI22], [JCRE22], [JCVM22].

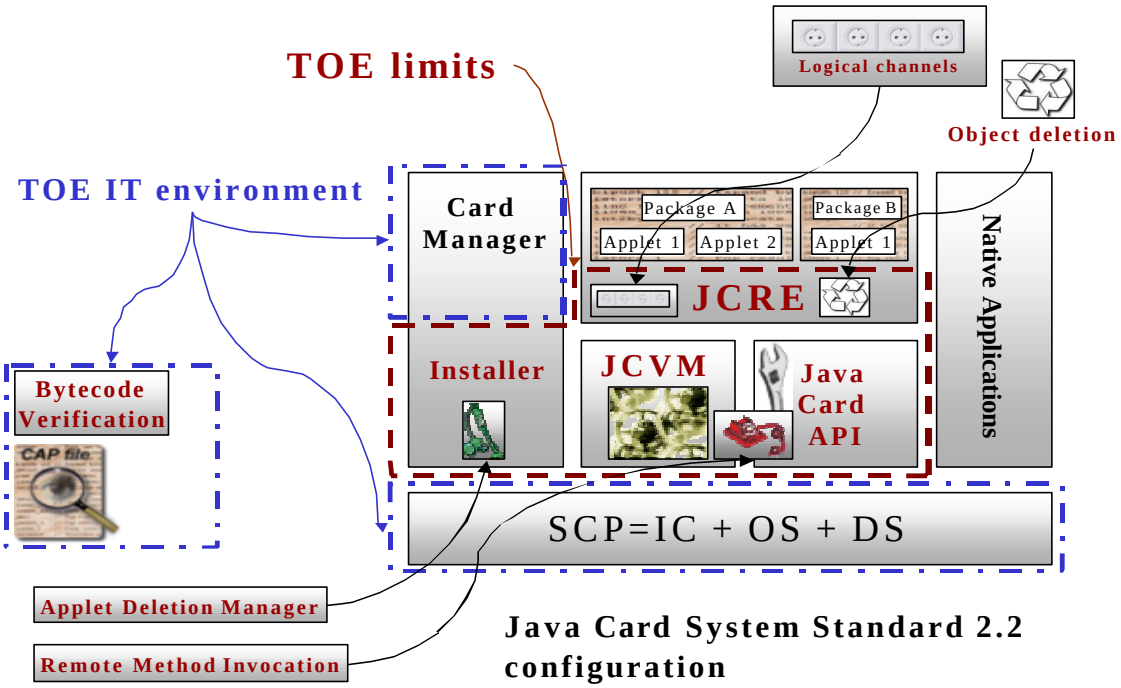

Figure 1 Sketch of the TOE

3.2 TOE IT ENVIRONMENT

The TOE IT Environment is composed of the following parts:

- the SCP (smart card platform)
- the card manager,
- the byte code verifier.

TOE IT environment is delimited with blue dashed line in Figure 1.

The smart card platform (SCP) is composed of a micro-controller and of a very thin software layer.

The Integrated Circuits Used in this product is: Samsung S3CJ9QD.

Samsung S3CJ9QD is a secure 32-bit risk CPU based on the core ARMv5TEJ specially designed for Smart Card and tailored to meet the highest security requirements of the Smart Card Market. It provides 256Kbytes of user ROM, 8Kbytes of RAM and 128K bytes of EEPROM.

The software layer is the part of ST-Incard Operating interfaceing with the Hardware. Basicaly it contains just the driver for the hardware resource management such as the I/O, compliant with ISO standards and the other hardware pheripheral. Moreover this softwarelayer contains drivers for crypto libraries.

Cryptolibrary are also provided by the chip manufacturer Samsung and are especially designed for the Hardware Coprocessor: Tornado™. The card manager is the application responsible for the administration of the smart card. The card manager is in charge of the life cycle of the whole card, as well as the installed applications. It is compliant with the GlobalPlatform Card Specification 2.1.1 [GP].

The byte code verifier is an off card tool used to perform static checks on the bytecodes of the methods in a CAP file. Bytecode verification is a key component of security: applet isolation, for instance, depends on the file satisfying the properties a verifier checks to hold. A method of a CAP file that has been verified, in addition, shall not contain, for instance, an instruction that allows to forge a memory address or an instruction that makes improper use of a return address as it were an object reference. In other words, bytecodes are verified to hold up to the intended use to which they are defined.

The bytecode verification includes:

- the well-formedness of the CAP file and the verification of the typing constraints on the bytecode;
- 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;
- bytecode instructions represent a legal set of Java Card instructions;
- adequacy of bytecode operands to bytecode semantics;
- absence of operand stack overflow/underflow;
- control flow confinement to the current method (that is, no control jumps to outside the method);
- absence of illegal data conversion and reference forging;
- enforcement of the private/public access modifiers for class and class members;
- 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);
- enforcement of rules for binary compatibility.

The bytecode verifier is the Java Card 2.2.1 Off-Card CAP File Verifier, Version 1.3

The Application Layer and the native applications are not part of the TOE and neither of the TOE it environment. However the security evaluation is performed considering the native applications.

The native applications included in this platform are the UICC, the USIM, the SIM respectively dedicated to 3G and 2G mobile telephony. Native application interact with the TOE as a java card applet by means of the same entry point methods.

3.3 THE TOE IN THE LIFE CYCLE OF THE SMART CARD

Following the CC, we separate the TOE environment into two parts: the IT environment and the non-IT environment. As seen in the preceding sections, the TOE is intended to be part of an IT product embedded in a smart card; due to specific development and installation processes of the smart card industry, these (the TOE's development and installation) are not separable from that of the other IT components of the smart card. This development phase constitutes the main part of the non-IT environment of the TOE.

The rest of this section is inspired by [PP0010], as we assume that JCRE is part of the embedded software (ES), so the same development rules shall apply. Note that [SCSUG-2] also presents an alternative (but less detailed) view of the development and production of smart card products.

The life cycle of the TOE, which is only a part of the smart card life cycle, can be reduced to the **three stages** pictured in Figure 2, called **Development**, **Production & Personalization**, and **Usage**.

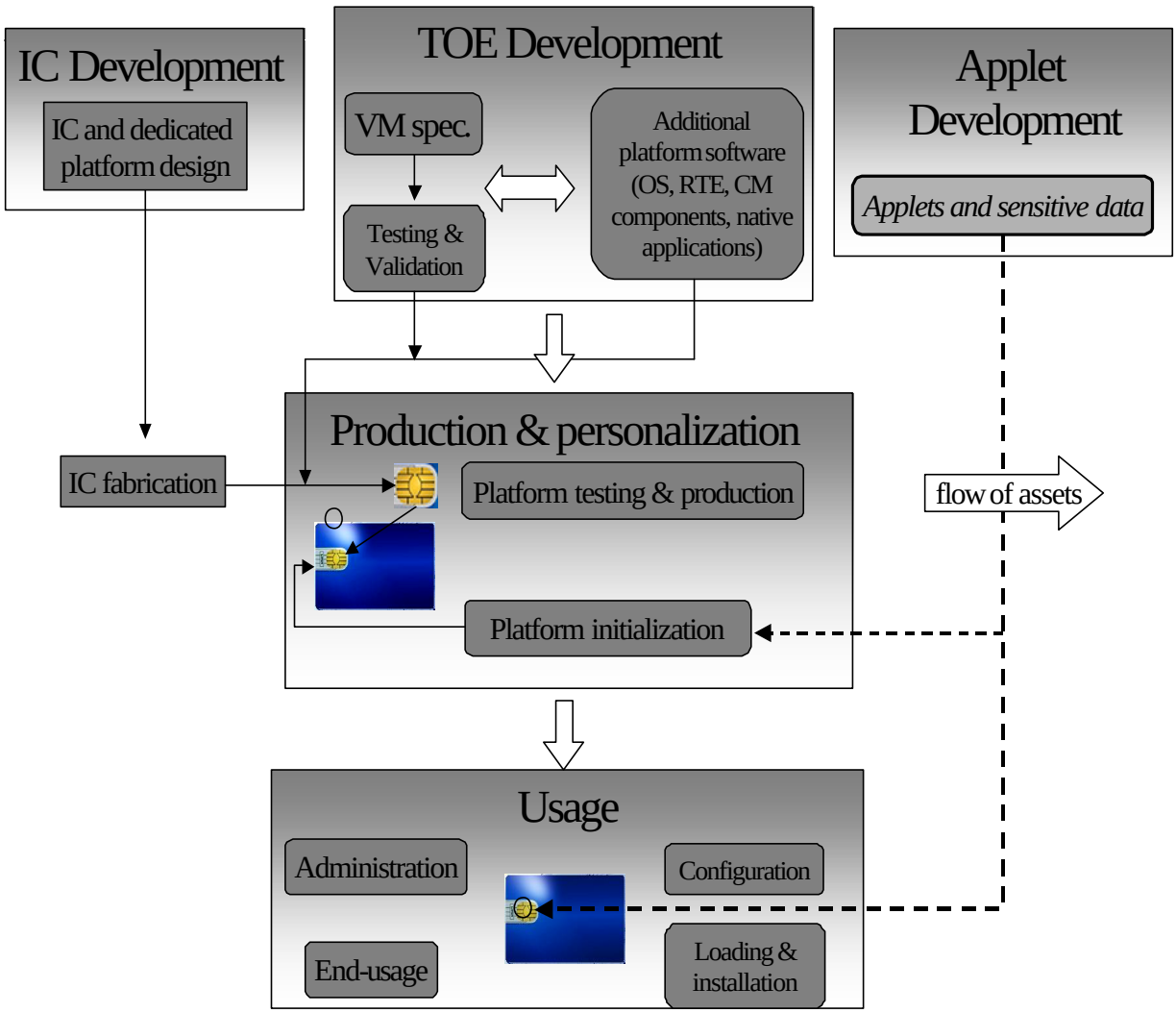

Figure 2: Smart Card Product Life Cycle

3.3.1 TOE Development & Production Environments

The development and production of the TOE is carried out during the first and second stages. To ensure security, the environment in which the development takes place must be made secure with controllable accesses and traceability. Furthermore, it is important that every authorized personnel involved fully understands the importance and the rigid implementation of defined security procedures.

The development begins with the TOE specification. All parties in contact with sensitive information are required to abide by Non-Disclosure Agreements.

Development of the TOE then follows. The engineers use a secure computer system (preventing unauthorized access) to make their specifications, design, development and generation of the product. Storage of sensitive documents, databases on tapes, diskettes are in appropriately locked cupboards/safe. The disposal of unwanted data (complete electronic erasures) and documents (like shredding) is also of great importance. Testing, integration and validation of TOE components then

take place. This phase consists in the collection of all software modules and the execution/test of this software on an emulator or on a simulator of the (DS & IC) layer.

When these are done offsite, they must be transported and worked out in a secure environment with accountability and traceability of all components. During the electronic transfer of sensitive data, procedures must be established to ensure that the data and programs reach the expected destination and are not accessible at intermediate stages (stored on a buffer server where system administrators make backup copies). Should the integration tests be successful, the ROM code is delivered to the IC manufacturer.

During **the production** stage the TOE is used in the IC Packaging, smart card Finishing process and the test environments. Everyone involved in such operations shall fully understand the importance of security procedures. Moreover, the environment in which these operations take place must be secured. Sensitive information (on tapes, disks or diskettes) is stored in an appropriately locked cupboard/safe. Also of paramount importance is the disposal of unwanted data (like complete electronic erasures) and documents (for instance, shredding). During production, the TOE is protected just like any other component of the smart card (SCP, test samples) and the smart card itself.

Personalization then occurs that is, the embedder introduces data for configuration and initialization of software components, namely the OS, the Java Card System, the SCP, and applications. At the end of the second stage, the TOE is fully functional. The initialization is part of the evaluation.

Adequate control procedures are necessary to account for all products at all stages. These must be transported and manipulated in a secure environment with accountability and traceability of all (good and bad) products.

3.3.2 TOE Final Environment

The **third stage** is the end usage time of the TOE.

Once the previous stage is over, the loading and installation of applications, and configuration (initialization) of user data (like user PIN) is done. The card is finally issued to the end user (card holder).

The main users of the TOE at this time are the applications, either pre-installed or loaded. The end user environment thus covers a wide spectrum of very different functions.

However, we can define the IT environment during this phase: first, the TOE obviously runs on top of what we called the SCP, and is itself part of the underlying platform for the card manager[1]. The underlying smart card platform has been described in §3.2. The TOE takes advantage of the features it provides for its own management needs, such as transaction facilities, memory management and safe cryptographic operations. At a lower level, the hardware provides physical protection of the TOE.

On the other side, the TOE communicates with the CAD through the card manager. The triumvirate made up of the JCRE, the installer and the *CM* is likely to be merged into one entity in actual implementations. However, each one is in charge of a distinct security role on which the separation is grounded.

[1] The card manager may also directly rely upon the SCP to access some of its low-level services.

During normal usage, the card is inserted in a CAD, starting up the CM and JCRE. The session is an exchange of APDU commands between the CAD and the CM, the CM and the JCRE and, ultimately, the JCRE and some applet.

Loading of an applet post-issuance follows the same pattern, with the exception that the JCRE hands over the reins to the installer for the duration of the procedure. It will get the control back when the newly loaded applet will need to be installed (that is, on the invocation of its **`install()`** method).

Finally, that loading issue leads us to another entity, which appears in Figure 1, the CAP file verifier (also known as "bytecode verifier", or, shortly, the BCV). The verifier can either be located off–card or on–card without loss of generality, although this choice is not necessarily innocuous to security issues (for instance, the integrity of the loaded file is important for off–card verification).

3.3.3 Roles Clarification

Following tables report a description of the TOE administrators and users.

Administrators	Description
Platform Developer	Develops the Embedded Software. The Embedded Software Developer of this product is ST-Incard.
Card Embedder	Is responsible of: • Manufacturing the smart card starting from the IC provided by the IC Manufacturer; • Initializing the smart card platform for the customer; • Loading and installing the applications on the card; • Personalizes the card with the customer' secret data. The Card Embedder of this product is ST-Incard.
Card Issuer	Issues the card to the end-user. The card belongs to the Card Issuer. The Card Issuer for this product could be a Telecom Operator.

Table 1 – TOE Administrators

Users	Description
Application Developer	Is in charge of designing and developing the application code. The Application Developer for this product is ST-Incard and/or other.
Card Holder	Customers of the Card Issuer.

Terminals	Mainly mobile equipments.

Table 2 – TOE Users

3.3.4 TOE Delivery and Deliverables

At the end of the "TOE development phase" the TOE is delivered to the chip manufacturer in the form of ROM code. A state of the TOE cannot be defined at this point sice the TOE is just a software image.

At the end of the production and personalization phase the TOE is delivered to the end users (see Table 2) on final products. The deliverables are smart card fully personalizedand and working, ready to provide services to end users. The TOE state is SECURED. This means that the TOE is working and the card has been equipped with all the applications and all the features requested by the Card Issuer. All the secret keys and data have been downloaded on card. The transition from other states to SECURED is irreversible.

The Administrator and User Guidance is also considered a deliverable.

The Administrator and User Guidance is delivered to:

- the Application Developers that are users during the Applet Development;
- the Card Embedder that is an administrator in the production and personalization phase;
- the Users and the Card Issuer that are user and admionistrator respectively in the usage phase.

3.3.5 Non-IT Environment

Following table reports a description of the TOE non-IT environments.

Environment	Description
Development	The environment of the phases "TOE Development" in the TOE lifecycle.
Production & Personalization	The environment of the phases "Production and Personalization" in the TOE lifecycle.
User	The environment of the phases "Usage" in the TOE lifecycle.

Table 3 – TOE non-IT environments

3.3.6 Lifecycle Rationale

The following table reports the TOE Lifecycle Rationale. Grey cases indicates the TOE evaluation phases.

The TOE life cycle can be decomposed roughly in 3 macro phases. Each phase has inputs which can be partially or completely the results of the previous phase and generates outputs (deliverables). In each phase of the TOE lifecycle there are user and administrators of the TOE and defines the TOE non-IT environment.

A description of the card states for each phase is reported in the Table 5.

Phase	Description	Deliverables	Environment	Administrators	Users	Card State
1	**TOE Development**	Embedded Software	Development	Platform Developer		None
	Applet Development	Application Software	Development		Application Developer	None
2	**Production & Personalization**	Card	Production	Card Embedder		PE_READY
3	User	Card	User	Card Issuer	Card Holder	SECURED

Table 4 – TOE Lifecycle Rationale

Card State	Description
TEST_MODE	The IC Manufacturer OS is active. The card just answers to the IC Manufacturer test commands.
PE_READY	The Card Manufacturer OS is Active. The card just answer to the OS APDU.
SECURED	The card is fully working. It has been equipped with all the applications and all the features requested by the Card Issuer. All the secret keys and data have been downloaded on card. The state transition from other states to SECURED is irreversible.

Table 5 – Card States

3.4 TOE INTENDED USAGE

Smart cards are mainly used as data carriers that are secure against forgery and tampering. More recent uses also propose them as personal, highly reliable, small size devices capable of replacing

paper transactions by electronic data processing. Data processing is performed by a piece of software embedded in the smart card chip, usually called an application.

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

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

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

So far, the most important applications are:

- Financial applications, like Credit/Debit ones, stored value purse, or electronic commerce, among others.
- Transport and ticketing, granting pre-paid access to a transport system like the metro and bus lines of a city.
- Telephony, through the subscriber identification module (SIM) for digital mobile telephones.
- Personal identification, for granting access to secured sites or providing identification credentials to participants of an event.
- 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.

The version 2.2 of the Java Card platform ("Java Card System 2.2") introduces several novelties that extend the domain of applications of the Java Card platform and ensures its compatibility with the industrial state-of-art standards. One of those features is the possibility of having more than one applet selected for execution at a time, which is intensively used in identity modules of mobile phone applications. A Java Card platform implementing this feature is said to support "logical channels".

Java Card System 2.2 also provides applet deletion, which enables the fine tuning of open card management. This typically impacts the loyalty applications, which are obvious candidates for post-issuance downloading and removal of applications.

Lastly, Java Card System 2.2 also provides support for object deletion and remote method invocation (RMI). Such features do not target any particular kind of applications. Object deletion enables the reallocation of memory blocks, while RMI services are intended to shrink the size of the applet code in charge of dispatching the commands received from the card host.

Applications can be loaded and installed both pre-issuance and post issuance.

Post issuance loading can be performed by means of the card manager or over the air (OTA) via 03.48 protocol if the TOE is used for mobile phone applications.

OTA post issuance loading via 03.48 is outside the limit of the ST.

4 TOE Security Environment

This chapter describes the security aspects of the environment in which the TOE is used. The first section describes some general security, and is intended to ease the comprehension of the security objectives and requirements, especially the access control policies. Sections §4.2 and §4.3 introduce the assets to be protected, the users of the TOE, and their software counterparts. Section §4.4 describes the assumptions made on the environment. Section §4.5 describes the threats menacing the assets of the TOE. Finally, the organizational policies that shall be imposed on the environment of the TOE are presented in Section §4.6.

4.1 SECURITY ASPECTS

Security aspects are intended to define the main security issues that are to be addressed in the ST, in a CC-independent way. In addition to this, they also give a semi-formal framework to express the CC security environment and objectives of the TOE. They can be instantiated as assumptions, threats, objectives (for the TOE and the environment), or organizational security policies. 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.

The meaning of this paragraph is to state that the TSFs must be continuously active in one way or another, and that aspect is termed "OPERATE". The ST may include an assumption, termed "A.OPERATE", stating that it is assumed that the TOE ensures continued correct operation of its security functions, and so on. But it may also include a threat, termed "T.OPERATE", to be interpreted as the negation of the statement #.OPERATE. In this example, this amounts to state that an attacker may try to circumvent some specific TSF by temporarily shutting it down. The use of a common name intends to ease the global understanding of the document.

This section presents several security aspects that will appear below in the configurations of the ST. Some being quite general, we give further details, which are numbered for easier cross-reference 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.

CONFIDENTIALITY

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

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

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

INTEGRITY

#.*INTEG-APPLI-CODE* Application code must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain write access to the memory zone where executable code is stored. Since the configuration allows post-issuance application loading, this threat also concerns the modification of application code in transit to the card.

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

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

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

UNAUTHORIZED EXECUTIONS

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

#.*EXE-JCS-CODE* Java Card System (byte)code must be protected against unauthorized execution. Java Card System (byte)code includes any code of the JCRE or API. This concerns **(1)** invoking a method outside the scope of the visibility rules provided by the public/private access modifiers of the Java

programming language ([JAVASPEC],§6.6); **(2)** jumping inside a method fragment or interpreting the contents of a data memory area as if it was executable code. Note that execute access to native code of the Java Card System and applications is the concern of #.*NATIVE*.

#.*FIREWALL* The Java Card System shall ensure controlled sharing of class instances[2], and isolation of their data and code between packages (that is, controlled execution contexts). **(1)** An applet shall neither read, write nor compare a piece of data belonging to an applet that is not in the same context, nor execute one of the methods of an applet in another context without its authorization.

#.*NATIVE* Because the execution of native code is outside of the TOE Scope Control (TSC), it must be secured so as to not provide ways to bypass the TSFs. No untrusted native code may reside on the card. Loading of native code, which is as well outside the TSC, is submitted to the same requirements. Should native software be privileged in this respect, exceptions to the policies must include a rationale for the new security framework they introduce.

BYTECODE VERIFICATION

#.*VERIFICATION* All bytecode must be verified prior to being executed. Bytecode verification includes **(1)** how well-formed CAP file is and the verification of the typing constraints on the bytecode, **(2)** binary compatibility with installed CAP 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.

CAP File Verification

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

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

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

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.

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

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.

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.

CARD MANAGEMENT

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

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

#.*SID* **(1)** Users and subjects of the TOE must be identified. **(2)** The identity of sensitive users and subjects associated with administrative and privileged roles must be particularly protected; this concerns the JCRE, the applets registered on the card, and especially the default applet and the currently selected applet (and all other active applets in Java Card System 2.2). A change of identity, especially standing for an administrative role (like an applet impersonating the JCRE), is a severe violation of the TOE Security Policy (TSP). Selection controls the access to any data exchange between

the TOE and the CAD and therefore, must be protected as well. The loading of a package or any exchange of data through the APDU buffer (which can be accessed by any applet) can lead to disclosure of keys, application code or data, and so on.

#*OBJ-DELETION*

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

#*DELETION*

Deletion of applets must be secure. **(1)** Deletion of installed 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 TSPs. This does not mandate, however, the process to be atomic. For instance, an interrupted deletion may result in the loss of user data, as long as it does not violate the TSPs.

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).

SERVICES

#*.ALARM*

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

#*.OPERATE*

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

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

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

4.2 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; 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 source code (one asset) or compiled code (another asset), and may exist in various formats (digital supports, printed paper) at different stages of its development. 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 weighs on it.

Each asset is named with prefix "*D.*" followed by the "*name*" identifying the data asset.

4.2.1 User data

D.APP_CODE	The code of the applets and libraries loaded on the card. To be protected from unauthorized modification.
D.APP_C_DATA	Confidential sensitive data of the applications, like the data contained in an object, a static field of a package, a local variable of the currently executed method, or a position of the operand stack. To be protected from unauthorized disclosure.
D.APP_I_DATA	Integrity sensitive data of the applications, like the data contained in an object, a static field of a package, a local variable of the currently executed method, or a position of the operand stack. To be protected from unauthorized modification.
D.PIN	Any end-user's PIN. To be protected from unauthorized disclosure and modification.

D.APP_KEYs Cryptographic keys owned by the applets.

To be protected from unauthorized disclosure and modification.

4.2.2 TSF data

D.JCS_CODE The code of the Java Card System.

To be protected from unauthorized disclosure and modification.

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

To be protected from monopolization and unauthorized disclosure or modification.

D.SEC_DATA The runtime security data of the JCRE, like, for instance, the 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 unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

4.3 USERS & SUBJECTS

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) 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 .

The main subjects of the TOE considered in this document are the following ones:

- Packages used on the Java Card platform that act on behalf of the applet developer. These subjects are involved in the ***FIREWALL security policy*** defined in §6.1.1.1 and they should be understood as instances of the subject ***S.PACKAGE***.

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

- The bytecode verifier (BCV), which acts on behalf of the verification authority. This subject is involved in the ***PACKAGE LOADING security policy*** defined in §6.1.7 and is represented by the subject ***S.BCV***.

- The installer, which acts on behalf of the card issuer. This subject is involved in the loading of packages and installation of applets. It could play the role of the on-card entity in charge of package loading, which is involved in the ***PACKAGE LOADING security policy*** defined in §6.1.7 and is represented by the subject ***S.CRD***.

- The applet deletion manager, if the configuration contains such components, which also acts on behalf of the card issuer. This subject is involved in the ***ADEL security policy*** defined in §6.1.3.1 and is represented by the subject ***S.ADEL***.

- The CAD is involved in the ***JCRMI security policy*** defined in §6.1.4.1 and is represented by the subject ***S.CAD***.

With the exception of packages, the other subjects have special privileges and play key roles in the security policies of the TOE.

A special subject is involved in the ***PACKAGE LOADING security policy,*** which acts as the entity that may potentially intercept, modify, or permute the messages exchanged between the verification authority and the on-card entity in charge of package loading.

4.4 ASSUMPTIONS

This section introduces the assumptions made on the environment of the TOE for each of the configurations considered in this document.

A.NATIVE Those parts of the APIs written in native code as well as any pre-issuance native application on the card are assumed to be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p.27) for details.

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.

A.APPLET applets loaded post-issuance do not contain native methods. The Java Card specification explicitly "does not include support for native methods" ([JCVM21], §3.3) outside the API.

4.5 THREATS

This section introduces the threats to the assets against which specific protection within the TOE or its environment is required. The classification is also inspired by the components of the TOE that are supposed to counter each threat.

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 DP analysis. That also includes the modification of the runtime execution of Java Card System or SCP software through alteration of the intended execution order of (set of) instructions through physical tampering techniques.

This threatens *all* the identified assets.

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

CONFIDENTIALITY

T.CONFID-JCS-CODE The attacker executes an application without authorization to disclose the Java Card System code. See #.*CONFID-JCS-CODE* (p. 25) for details.

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

T.CONFID-APPLI-DATA The attacker executes an application without authorization to disclose data belonging to another application. See #.*CONFID-APPLI-DATA* (p. 25) for details.

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

T.CONFID-JCS-DATA The attacker executes an application without authorization to disclose data belonging to the Java Card System. See #.*CONFID-JCS-DATA* (p. 26) for details.

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

INTEGRITY

T.INTEG-APPLI-CODE The attacker executes an application to alter (part of) its own or another application's code. See #.*INTEG-APPLI-CODE* (p. 26) for details.

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

T.INTEG-JCS-CODE The attacker executes an application to alter (part of) the Java Card System code. See #.*INTEG-JCS-CODE* (p. 26) for details.

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

T.INTEG-APPLI-DATA The attacker executes an application to alter (part of) another application's data. See #.*INTEG-APPLI-DATA* (p. 26) for details.

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

T.INTEG-JCS-DATA The attacker executes an application to alter (part of) Java Card System or API data. See #.*INTEG-JCS-DATA* (p. 26) for details.

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

T.INTEG-APPLI-CODE.2 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* (p. 26) for details.

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

T.INTEG-APPLI-DATA.2 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* (p. 26) for details.

Directly threatened asset(s): **D.APP_I_DATA, D_APP_KEYs** and **D.JCS_KEYs**.

IDENTITY USURPATION

T.SID.1 An applet impersonates another application, or even the JCRE, in order to gain illegal access to some resources of the card or with respect to the end user or the terminal. See #.*SID* (p. *28*) for details.

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

T.SID.2 The attacker modifies the identity of the privileged roles. See #.*SID* (p. *28*) 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).

UNAUTHORIZED EXECUTION

T.EXE-CODE.1 An applet performs an unauthorized **execution** of a **method**. See #.*EXE-JCS-CODE* (p. 26) and #.*EXE-APPLI-CODE* (p. *26*) for details.

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

T.EXE-CODE.2 An applet performs an unauthorized **execution** of a **method fragment** or **arbitrary data**. See #.*EXE-JCS-CODE* (p. *26*) and #.*EXE-APPLI-CODE* (p. *26*) for details.

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

T.NATIVE An applet **executes** a **native method** to bypass a security function such as the firewall. See #.*NATIVE* (p. 27) for details.

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

DENIAL OF SERVICE

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

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

MODIFICATIONS OF THE SET OF APPLICATIONS

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* (p *28*) 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).

UNAUTHORIZED EXECUTIONS

T.EXE-CODE-REMOTE The attacker performs an unauthorized **remote execution** of a **method** from the CAD. See #.*EXE-JCS-CODE* (p. 26) and #.*EXE-APPLI-CODE* (p. *26*) for details.

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

This threat concerns version 2.2 of the Java Card System remote method invocation features, 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 TSC.

CARD MANAGEMENT

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* (p *29*) for details).

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

SERVICES

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* (p. 29) for further details.

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

4.6 ORGANIZATIONAL SECURITY POLICIES

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

OSP.VERIFICATION This policy shall ensure the adequacy 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* (p.*27*) for details.

5 SECURITY OBJECTIVES

This section defines the security objectives to be achieved by each of the TOE configurations considered in this document and their respective environments.

5.1 SECURITY OBJECTIVES FOR THE TOE

IDENTIFICATION

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

EXECUTION

O.OPERATE The TOE must ensure continued correct operation of its security functions. See #.*OPERATE* (p *29*) for details.

O.RESOURCES The TOE shall control the availability of resources for the applications. See #.*RESOURCES* (p *30*) for details.

O.FIREWALL The TOE shall ensure controlled sharing of data containers owned by applets of different packages, and between applets and the TSFs. See #.*FIREWALL* (p *27*) for details.

O.NATIVE The only means that the JCVM shall provide for an application to execute native code is the invocation of a method of the Java Card API, or any additional API. See #.*NATIVE* (p *27*) for details.

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

Application note: To be made unavailable means to be physically erased with a default value. Except for local variables that do not correspond to method parameters, the default values to be used are specified in [JCVM21].

O.SHRD_VAR_CONFID The TOE shall ensure that any data container that is shared by all applications is always cleaned after the execution of an application. Examples of such shared containers are the APDU buffer, the byte array

used for the invocation of the `process` method of the selected applet, or any public global variable exported by the API.

O.SHRD_VAR_INTEG The TOE shall ensure that only the currently selected application may grant write access to a data memory area that is shared by all applications, like the APDU buffer, the byte array used for the invocation of the `process` method of the selected applet, or any public global variable exported by the API. Even though the memory area is shared by all applications, the TOE shall restrict the possibility of getting a reference to such memory area to the application that has been selected for execution. The selected application may decide to temporarily hand over the reference to other applications at its own risk, but the TOE shall prevent those applications from storing the reference as part of their persistent states.

SERVICES

O.ALARM The TOE shall provide appropriate feedback information upon detection of a potential security violation. See #.*ALARM* (p. *29*) for details.

O.TRANSACTION The TOE must provide a means to execute a set of operations atomically. See #.*TRANSACTION* (p. *31*) for details.

O.CIPHER The TOE shall provide a means to cipher sensitive data for applications in a secure way. In particular, the TOE must support cryptographic algorithms consistent with cryptographic usage policies and standards. See #.*CIPHER* (p. *30*) for details.

O.PIN-MNGT The TOE shall provide a means to securely manage PIN objects. See #.*PIN-MNGT* (p. *30*) for details.

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

O.KEY-MNGT The TOE shall provide a means to securely manage cryptographic keys. This concerns the correct generation, distribution, access and destruction of cryptographic keys. See #.*KEY-MNGT* (p. *30*).

Application note: *O.KEY-MNGT*, *O.PIN-MNGT*, *O.TRANSACTION* and *O.CIPHER* are actually provided to applets in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to applets; those may be built on top of the Java Card API or independently. Depending on whether they contain native code or not, these proprietary libraries will need to be evaluated together with the TOE or not (see #.*NATIVE*, p.27). In any case, they are not included in the Java Card System for the purpose of the present document.

APPLET MANAGEMENT

O.INSTALL The TOE shall ensure that the installation of an applet is safe. See #.*INSTALL* (p *28* for details).

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

Application note: Usurpation of identity resulting from a malicious installation of an applet on the card may also be the result of perturbing the communication channel linking the CAD and the card. Even if the CAD is placed in a secure environment, the attacker may try to capture, duplicate, permute or modify the packages sent to the card. He may also try to send one of its own applications as if it came from the card issuer . Thus, this objective is intended to ensure the integrity and authenticity of loaded CAP files.

O.DELETION The TOE shall ensure that both applet and package deletion are safe. See #.*DELETION* (p *29*) for details.

OBJECT DELETION

O.OBJ-DELETION The TOE shall ensure the object deletion shall not break references to objects. See #.*OBJ-DELETION* (p. 29) for further details.

SERVICES

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

5.2 SECURITY OBJECTIVES FOR THE ENVIRONMENT

This section introduces the security objectives to be achieved by the environment associated to each TOE configuration.

OE.NATIVE Those parts of the APIs written in native code as well as any pre-issuance native application on the card shall be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p.27) for details.

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

OE.SCP.SUPPORT The SCP shall provide functionalities that support the well-functioning of the TSFs of the TOE (avoiding they are bypassed or altered) and by controlling the access to information proper of the TSFs. In addition, the smart card platform should also provide basic services which are required by the runtime environmernt to implement security mechanisms such as atomic transactions, management of persistent and transient objects and cryptographic functions. These mechanisms are likely to be used by security functions implementing the security requirements defined for the TOE. See #.*SCP.2-5* (p.30).

OE.SCP.IC The SCP shall possess IC security features. See #.*SCP.7* (p.*30*).

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.

OE.VERIFICATION All the bytecodes shall be verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time. See #.*VERIFICATION* (p.*27*) for details.

OE.APPLET No applet loaded post-issuance shall contain native methods.

6 IT SECURITY REQUIREMENTS

In this chapter the list of security functional requirements is reported. Security functional requirements were divided in groups on the basis of the functional component they refer to. Here follows a description of each group of requirements and the location with respect to the TOE (TOE or IT security environment).

The minimum strength level for the TOE security functions is SOF-High.

Group	Group Name	Applied to	Description
Core group	CoreG	TOE	Contains the basic requirements concerning the runtime environment of the Java Card System, such as the firewall policy and the requirements related to the Java Card API.
SCP group	SCPG	IT	Contains the security requirements for the smart card platform, that is, operating system and chip that the Java Card System is implemented upon. It does not define requirements for the TOE but for its IT environment.
Installation group	InstG	TOE	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.
RMI group	RMIG	TOE	Contains the security requirements for the remote method invocation features, which provides a new protocol of communication between the terminal and the applets. This was introduced in Java Card System 2.2.
Logical channels group	LCG	TOE	Contains the security requirements for the logical channels, which provide a runtime environment where several applets can be simultaneously selected or a single one can be selected more than once. This is a Java Card System 2.2 feature.
Object deletion group	ODELG	TOE	Contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card System 2.2 feature.
Bytecode verification group	BCVG	IT	Contains the security requirements concerning the bytecode verification of the application code to be loaded on the card. This group of SFRs may apply to the TOE or to its IT environment depending on the configuration.
Applet deletion group	ADELG	TOE	Contains the security requirements for erasing installed applets from the card, a new feature introduced in Java Card System 2.2. It can also be used as a basis for any other application deletion requirements.
Secure carrier group	CarG	TOE	Contains minimal requirements for secure downloading of applications on the card. This group contains the security requirements for preventing, in those configurations which do not support on-card static or dynamic verification of bytecodes, the installation of a package that has not been bytecode verified, or that has been modified after bytecode verification.
Card manager group	CMGRG	IT	Contains the minimal requirements that allow defining a policy for controlling access to card content management operations and for expressing card issuer security concerns.

Table 6 - Group of Requirement and target of application

6.1 TOE SECURITY FUNCTIONAL REQUIREMENTS

6.1.1 CoreG Security Functional Requirements

This group is focused on the main security policy of the Java Card System, known as the firewall. This policy essentially concerns the security of *installed* applets, along with a small part that relates to the installation procedure. The policy focuses on the execution of bytecodes.

As reported in the [JCSPPC] at §5.1.6 and in the table 27 at §7 some security requirements concerning the CoreG group shall be updated to address the security requirements introduced in the configuration Java Card Standard 2.2 with the LCG group. This is clearly stated in [JCSPPC] at §5.1.6 and even more clearly in in the table 27 at §7.

The LCG Security Functional Requirements introduces a reformulation of the ***FIREWALL SFP*** specified in the group CoreG Security Functional Requirements (*FDP_ACF.1.1/FIREWALL, FDP_ACF.1.2/FIREWALL, FMT_MSA.1.1/JCRE)* and a modification to a component of the security requirement for residual information protection (*FDP_RIP1.1/TRANSIET*). The modification of this policy has been highlithed in yellow.

The security issues introduced by logical channels are mainly related to the access to SIO objects owned by legacy applets as well as to the clearing of transient data which is shared by applet instances which are concurrently active in different logical channels.

6.1.1.1 Firewall Policy

FDP_ACC.2: COMPLETE ACCESS CONTROL

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

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

Subject	Description
S.PACKAGE	Any package, which is the security unit of the firewall policy.
S.JCRE	The JCRE. This is the process that manages applet selection and de-selection, along with the delivery of APDUs from and to the smart card device. *This subject is unique.*

Subject	Description
O.JAVAOBJECT	Any object. Note that KEYS, PIN, arrays and applet instances are specific objects in the Java programming language.

Table 7 – Subject and Object of the firewall SFP

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

Operation	Description
OP.ARRAY_ACCESS(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, arg_1,… *)*	Invoke a virtual method (either on a class instance or an array object)
OP.INVK_INTERFACE(O.JAVAOBJECT, method, arg_1,… *)*	Invoke an interface method.
OP.THROW(O.JAVAOBJECT)	Throwing of an object (**athrow**).
OP.TYPE_ACCESS(O.JAVAOBJECT, class*)*	Invoke **checkcast** or **instanceof** on an object.
OP.JAVA(…)	Any access in the sense of [JCRE21], §6.2.8. In our formalization, this is one of the preceding operations.
*OP.CREATE(*Sharing, LifeTime*)*	Creation of an object (**new** or **makeTransient** call).

Table 8 – Operations among subjects and objects covered by the firewall SFP

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

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

See ***FMT_MSA.1*** for more information about security attributes.

FDP_ACF.1.1/FIREWALL The TSF shall enforce the ***FIREWALL access control SFP*** to objects based on the following: ***(1) the security attributes of the covered subjects and objects, (2) the currently active context, (3) the SELECTed applet Context, and (4) the attribute ActiveApplets, which is a list of the active applets' AIDs.***

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

Subject/Object	Attributes
S.PACKAGE	Context, Selection Status
S.JCRE	None
O.JAVAOBJECT	Sharing, Context, LifeTime

Table 9 – Security Attribute of the firewall SFP

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

Name	Description
Context	Package AID, or "JCRE"
Sharing	Standard, SIO, JCRE entry point, or global array
LifeTime	**CLEAR_ON_DESELECT** or **PERSISTENT**.[3]
SELECTed applet Context	Package AID, or "None"
Selection Status	Multiselectable, Non-multiselectable or "None"
ActiveApplets	List of package's AIDs

Table 10 – Security attribute values of the firewall SFP

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

[3] 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.

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 ([JCRE21], §6.1.2). An object is owned by an applet instance, by the JCRE or by the package library where it has been defined (these latter objects can only be arrays that initialize static fields of packages).

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

> ([JCRE21], Glossary) Currently selected applet. The JCRE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command with this applet's AID, the JCRE makes this applet the currently selected applet. The JCRE sends all APDU commands to the currently selected applet.

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

> ([JCRE21] §6.1.1) At any point in time, there is only **one active context** within the VM (this is called the *currently active context*).

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

The Java Card platform, version 2.2, 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 being selected simultaneously. These applets are referred to as *multiselectable* applets. Applets that belong to a same package are either all multiselectable or not ([JCVM22],§2.2.5). Therefore, the selection mode can be regarded as an attribute of packages. No selection mode is defined for a library package.

Support for multiple logical channels (with multiple selected applet instances) requires a change to the Java Card System, version 2.1.1, concept of *selected applet*. Since more than one applet instance can be selected at the same time, and one applet instance can be selected on different logical channels simultaneously, it is necessary to differentiate the state of the applet instances in more detail. An applet instance will be considered an *active applet instance* if it is currently selected in at least one logical channel, up to a maximum of four. 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).

The ActiveApplets security attribute is internal to the VM, that is, not attached to any specific object or subject of the SPM. The attribute is TSF data that plays a role in the SPM.

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

R.JAVA.1 ([JCRE21]§6.2.8) An ***S.PACKAGE*** may freely perform any of ***OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW*** or ***OP.TYPE_ACCESS*** upon any ***O.JAVAOBJECT*** whose Sharing attribute has value "JCRE entry point" or **"global array".**

R.JAVA.2 ([JCRE21]§6.2.8) An ***S.PACKAGE*** may freely perform any of ***OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE*** or ***OP.THROW*** upon any ***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 ([JCRE21]§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 ([JCRE22], §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 one of the following 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,

and in either of the cases above the invoked interface method extends the **`Shareable`** interface.

R.JAVA.5 An *S.PACKAGE* may perform an *OP.CREATE* only if the value of the Sharing parameter[4] is "Standard".

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

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

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

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

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

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

FDP_IFC.1 SUBSET INFORMATION FLOW CONTROL

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

Subjects[5] (prefixed with an "*S*") and information (prefixed with an "*I*") covered by this policy are:

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

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

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

Table 11 – Subject and Information of the firewall SFP

There is a unique operation in this policy:

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

Table 12 – Operations on information of the firewall SFP

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

FDP_IFF.1 SIMPLE SECURITY ATTRIBUTES

FDP_IFF.1.1/JCVM The TSF shall enforce the ***JCVM information flow control SFP*** based on the following types of subject and information security attributes: ***(1) the currently active context.***

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

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

FDP_IFF.1.3/JCVM The TSF shall enforce ***the following additional information flow control: none***

FDP_IFF.1.4/JCVM The TSF shall provide the following a***dditionalSFP capabilities: none.***

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

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

Application note: the storage of temporary JCRE-owned objects' references is runtime-enforced ([JCRE21], §6.2.8.1-3).

Note that this policy essentially applies to the execution of bytecode. Native methods, the JCRE itself and possibly some API methods can be granted specific rights or limitations through the ***FDP_IFF.1.3/JCVM*** to ***FDP_IFF.1.6/JCVM*** elements.

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

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

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

Application note: The modification of the active context, SELECTed applet Context and ActiveApplets security attributes should be performed in accordance with the rules given in [JCRE22], §4 ***and ([JCVM22], §3.4.***.

FMT_MSA.2 SECURE SECURITY ATTRIBUTES

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

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

- The Context attribute of a ***.JAVAOBJECT**[6] must correspond to that of an installed applet or be "JCRE".
- An ***O.JAVAOBJECT*** whose Sharing attribute is a JCRE entry point or a global array necessarily has "JCRE" as the value for its Context security attribute.
- An ***O.JAVAOBJECT*** whose Sharing attribute value is a global array necessarily has "array of primitive Java Card System type" as a JavaCardClass security attribute's value.
- Any ***O.JAVAOBJECT*** whose Sharing attribute value is not "Standard" has a **PERSISTENT**-LifeTime attribute's value.
- Any ***O.JAVAOBJECT*** whose LifeTime attribute value is not **PERSISTENT** has an array type as JavaCardClass attribute's value.

Application note: The last two rules are motivated by the fact that the Java Card API defines only transient arrays factory methods.

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

Application note: Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (*FMT_MSA.1/JCRE*). At the creation of an object (***OP.CREATE***), the newly created object, assuming that the operation is permitted by the SFP, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([JCRE21], §6.1.2). There is one default value for the *SELECTed applet Context* that is the *default applet identifier's Context*, and one default value for the *active context*, that is "*JCRE*".

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

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

Application note: The intent is that none of the identified roles has privileges with regard to the default values of the security attributes. Notice that creation of objects is an operation controlled by the ***FIREWALL SFP***; the latitude on the parameters of this operation is described there. The operation shall fail

[6] Either subject or object.

anyway if the created object would have had security attributes whose value violates ***FMT_MSA.2.1/JCRE***.

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/JCRE The TSF shall be capable of performing the following security management functions: ***modification of the list of registered applets' AID, modification of the active context, modification of the SELECTed applet context security attributes and modification of the ActiveApplets security attributes***.

Application note: Added because [CCFI_065].

FMT_SMR.1 SECURITY ROLES

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

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

FPT_SEP.1 TSF DOMAIN SEPARATION

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

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

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

6.1.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

FCS_CKM.1 CRYPTOGRAPHIC KEY GENERATION

FCS_CKM.1.1/RSA The TSF shall generate cryptographic KEYS in accordance with a specified cryptographic KEY generation algorithm ***(RSA)*** and specified cryptographic KEY sizes: ***512 bits, 768 bits, 1024 bits*** that meet the following: ***[JCAPI22].***

FCS_CKM.1.1/SCP02 The TSF shall generate cryptographic KEYS in accordance with a specified cryptographic KEY generation algorithm ***DES Session Keys*** and specified cryptographic KEY sizes: ***128 bits*** that meet the following: [GP].

Application note: The keys can be generated and diversified in accordance with [JCAPI21] specification in classes **KeyBuilder** and **KeyPair** (at least Session key generation).

FCS_CKM.2 CRYPTOGRAPHIC KEY DISTRIBUTION

FCS_CKM.2.1/DES The TSF shall distribute cryptographic KEYS in accordance with a specified cryptographic KEY distribution method ***setKey()*** *of the interface* ***javacard.security.DESKey*** that meets the following ***standard: [JCAPI22].***

FCS_CKM.2.1/AES The TSF shall distribute cryptographic KEYS in accordance with a specified cryptographic KEY distribution method ***setKey()****of the interface* ***javacard.security.AESKey*** that meets the following ***standard: [JCAPI22].***

FCS_CKM.2.1/RSA STD The TSF shall distribute cryptographic KEYS in accordance with a specified cryptographic KEY distribution method ***setExponent()*** *and* ***setModulus()*** *of the interfaces* ***javacard.security.RSAPrivateKey*** *and* ***javacard.security.RSAPrivateKey*** that meets the following ***standard: [JCAPI22].***

FCS_CKM.2.1/RSA CRT The TSF shall distribute cryptographic KEYS in accordance with a specified cryptographic KEY distribution method ***setP(), setQ(), setPQ(), setDP1() , setDQ1,*** *of the interfaces* ***javacard.security.RSAPrivateCrtKey*** that meets the following: ***[JCAPI22].***

FCS_CKM.3 CRYPTOGRAPHIC KEY ACCESS

FCS_CKM.3.1/DES The TSF shall performe ***cryptographic key escrow*** in accordance with a specified cryptographic KEY access method ***getKey()*** *of the interface* ***javacard.security.DESKey*** that meets the following ***standard: [JCAPI22].***

FCS_CKM.3.1/AES The TSF shall performe ***cryptographic key escrow*** in accordance with a specified cryptographic KEY access method ***getKey()*** *of the interface* ***javacard.security.AESKey*** that meets the following ***standard: [JCAPI22].***

FCS_CKM.3.1/RSA STD The TSF shall performe ***cryptographic key escrow*** in accordance with a specified cryptographic KEY access method ***getExponent()*** *and*

`getModulus()` of the interfaces `javacard.security.RSAPrivateKey` and `javacard.security.RSAPrivateKey` that meets the following ***standard: [JCAPI22].***

FCS_CKM.3.1/RSA CRT The TSF shall performe ***cryptographic key escrow*** in accordance with a specified cryptographic KEY access method ***`getP(),getQ(),getPQ(),getDP1(),getDQ1(),` of the interfaces `javacard.security.RSAPrivateCrtKey`*** that meets the following: ***[JCAPI22].***

FCS_CKM.4 CRYPTOGRAPHIC KEY DESTRUCTION

FCS_CKM.4.1 The TSF shall destroy cryptographic KEYS in accordance with a specified cryptographic KEY destruction method ***`clearKey()` of the `interface javacard.security.Key`*** that meets the following ***standard : [JCAPI22].***

Application note: The keys are reset in accordance with [JCAPI21] in class **`Key`** with the method **`clearKey()`**. Any access to a cleared key attempting to use it for ciphering or signing shall throw an exception.

FCS_COP.1 CRYPTOGRAPHIC OPERATION

FCS_COP.1.1/DES The TSF shall perform ***encryption and decryption operations*** in accordance with a specified cryptographic algorithm ***Data Encryption Standard (DES)*** and cryptographic KEY sizes of ***64 bits, 128 bits and 192 bits*** that meet the following standard: ***U.S. Departement of Commerce/National Bureau of Standards Data Encryption Standard (DES), FIPS PUB 46-3, 1999 October 25.***

FCS_COP.1.1/AES The TSF shall perform ***encryption and decryption operations*** in accordance with a specified cryptographic algorithm ***(AES)*** and cryptographic KEY sizes ***128 bits, 192 bits, 256 bits*** that meet the following standard: ***Specification for the Advanced Encryption Standard FIPS 197 November 26, 2001***.

FCS_COP.1.1/RSA The ***TSF*** shall perform ***encryption and decryption operations in accordance with a specified cryptographic algorithm (RSA) and cryptographic KEY sizes 512 bits, 768 bits, 1024 bits that meet the following: PKCS#1: RSA Encryption Standard, Version 1.5, RSA Laboratories.The TOE shall provide a subset of cryptographic operations defined in [JCAPI21] in accordance to [JCAPI21] specification (see*** *javacardx.crypto.Cipher* ***and*** *javacardx.security* ***packages).***

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

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

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

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

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

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

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

Application note: The clearing of **`CLEAR_ON_DESELECT`** objects is not necessarily performed when the owner of the objects is deselected. In the presence of multiselectable `applet` instances, **`CLEAR_ON_DESELECT`** memory segments may be attached to `applet`s that are active in different logical channels. Multiselectable `applet` instances within a same `package` must share the transient memory segment if they are concurrently active ([JCRE22], *§4.2*.

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

Application note: The events that provoke the de-allocation of the previously mentioned references are described in [JCRE21], §7.6.3.

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

Application note: The **`javacard.security`** & **`javacardx.crypto`** `package`s do provide secure interfaces to the ***cryptographic buffer*** in a transparent way. See **`javacard.security.KeyBuilder`** and **`Key`** `interface` of [JCAPI21].

Application note: Java Card System 2.1.1 defines no explicit (or implicit) de-allocation of objects, but those caused by the failure of installation or the abortion of a transaction. The only related function for keys is the **`clearKey()`**

method, which does not mandate erasure of the contents of the key (see ***FCS_CKM.4***) nor the behavior of the transaction with respect to this "clearing". ST authors may consider additional security requirements on this topic.

FDP_ROL.1 BASIC ROLLBACK

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

FDP_ROL.1.2/FIREWALL The TSF shall permit operations to be rolled back within the ***scope of a*** **`select()`**, **`deselect()`**, **`process()`** or **`install()`** ***call, notwithstanding the restrictions given in*** [JCRE21], §7.7, within the bounds of ***the Commit Capacity*** ([JCRE21], §7.8), ***and those described in*** [JCAPI21].

Application note: Transactions are a service offered by the APIs to applets. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [JCAPI21] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).

Application note: The loading and linking of applet packages (the installation or registration is covered by ***FDP_ROL.1.1/FIREWALL***) is subject to some kind of rollback mechanism (see ***FPT_RCV.3.1/Installer***), described in [JCRE21], §10.1.4, but is implementation-dependent.

6.1.1.3 Card Security Management

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

FAU_ARP.1 SECURITY ALARMS

FAU_ARP.1.1/JCS The TSF shall ***throw an exception, lock the card session or reinitialize the Java Card System and its data or reject the CAP file*** upon detection of a potential security violation.

REFINEMENT Potential security violation is refined to 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
- Abortion of a transaction in an unexpected context (see (**abortTransaction()**, [JCAPI21] and ([JCRE21], §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)

Application note: The thrown exceptions and their related events are described in [JCRE21], [JCAPI21], and [JCVM21].

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

FDP_SDI.2 STORED DATA INTEGRITY MONITORING AND ACTION

FDP_SDI.2.1/PIN The TSF shall monitor user data stored within the TSC for ***integrity errors*** on all objects, based on the following attributes:

- ***D.PIN value***
- ***D.PIN object***

FDP_SDI.2.2/PIN Upon detection of a data integrity error, the TSF ***shall deny the use of the corrupted key an thrown an exception.***

FDP_SDI.2.1/KEY The TSF shall monitor user data stored within the TSC for ***integrity errors*** on all objects, based on the following attributes:

- ***D. APP_KEYs value***
- ***D.APP_KEYs object***

FDP_SDI.2.2/KEY Upon detection of a data integrity error, the TSF shall deny the use of the corrupted key an thrown an exception.

Application note: Although no such requirement is mandatory in the specification, at least an exception shall be raised upon integrity errors detection on cryptographic keys, PIN values and their associated security attributes. Even if all the objects cannot be monitored, cryptographic keys and PIN objects shall be considered with particular attention as they play a key role in the overall security.

FPT_RVM.1 NON-BYPASSABILITY OF THE TSP

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

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

FPT_TDC.1 INTER-TSF BASIC TSF DATA CONSISTENCY

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

Application note: Concerning the interpretation of data between the TOE and the underlying Java Card platform, it is assumed that the TOE is developed consistently with the SCP functions, namely concerning memory management, I/O functions, cryptographic functions, and so on.

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

- ***The*** [JCVM21] ***specification;***
- ***Reference export files;***
- ***The ISO 7816-6 rules;***
- ***The EMV specification***

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

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

Application note: The JCRE Context is the Current context when the VM begins running after a card reset ([JCRE21], §6.2.3). Behavior of the TOE on power loss and reset is described in [JCRE21], §3.5, and §7.1.

FPR_UNO.1 UNOBSERVABILITY

FPR_UNO.1.1/PIN The TSF shall ensure that ***any user*** is unable to observe the operations ***of comparisons*** on ***D.PIN*** by ***any subject***

FPR_UNO.1.1/KEY The TSF shall ensure that ***any user*** is unable to observe the ***cryptographic*** operations on ***D.APP_KEYs*** by ***any subject.***

Application note: Although it is not required in [JCRE21] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart card world. The precise list of operations and objects is left unspecified in the PP/0305, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.

FPT_TST.1 TSF TESTING

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

Application note: TSF-testing is not mandatory in [JCRE21], but appears in most of security requirements documents for masked applications.

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

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

6.1.1.4 AID Management

FMT_MTD.1 MANAGEMENT OF TSF DATA

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

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

FMT_MTD.3 SECURE TSF DATA

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

FIA_ATD.1 USER ATTRIBUTE DEFINITION

FIA_ATD.1.1/AID The TSF shall maintain the following list of security attributes belonging to individual users: ***the AID and version number of each package, the AID of each registered applet, and whether a registered applet is currently selected for execution ([JCVM21], §6.5).***

FIA_UID.2 USER IDENTIFICATION BEFORE ANY ACTION

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

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

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

FIA_USB.1 USER-SUBJECT BINDING

FIA_USB.1.1 The TSF shall associate the appropriate user security attributes with subjects acting on behalf of that user.

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

6.1.2 InstG Security Functional Requirements

This group bulks the SFRs related to the installation of the applets, which addresses security aspects outside the runtime. The idea here is that installation of applets is a critical phase, which lies partially out of the boundaries of the firewall, and therefore has to be deserved specific treatment. In the Common Criteria model, loading a package or installing an applet was considered as being an 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).

See also ***FIA_ATD.1***, ***FIA_USB.1***, ***FMT_MTD.1***, ***FMT_SMR.1*** for various information about applet installation.

FDP_ITC.2 IMPORT OF USER DATA WITH SECURITY ATTRIBUTES

FDP_ITC.2.1/Installer The TSF shall enforce the ***FIREWALL access control SFP*** when importing user data, controlled under the SFP, from outside of the TSC.

Application note: The most common importation of user data is package loading and applet installation on the behalf of the installer. Security attributes consist of the shareable flag of the class component, AID and version numbers of the package, maximal operand stack size and number of local variables for each method, and export and import components (visibility).

FDP_ITC.2.2/Installer The TSF shall use the security attributes associated with the imported user data.

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.

Application note: The format of the CAP file is precisely defined in Sun's specification ([JCVM21]); it contains the user data (like applet's code and data) and the security attribute altogether. Therefore there is no association to be carried out elsewhere.

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.

Application note: Each package contains a package Version attribute, which is a pair of major and minor version numbers ([JCVM21], §4.5). With the AID, it describes the package defined in the CAP file. When an export file is used during preparation of a CAP file, the versions numbers and AIDs indicated in the export file are recorded in the CAP files ([JCVM21], §4.5.2): the dependent packages Versions and AIDs attributes allow the retrieval of these identifications. However, package files do have "package Version Numbers" ([JCVM21]) used to indicate binary compatibility or incompatibility between successive implementations of a package, which obviously directly concern this requirement.

FDP_ITC.2.5/Installer The TSF shall enforce the following rule when importing user data controlled under the SFP from outside the TSC:

A package may depend on (import or use data from) other packages already installed. This dependency is explicitly stated in the loaded package in the form of a list of package AIDs. The 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 former is equal (less than or equal) to the major (minor) Version attribute associated to the latter ([JCVM21],§4.5.2). The intent of this rule is to ensure the binary compatibility of the package with those already on the card ([JCVM21], §4.4).

Application note: The installation (the invocation of an applet's **install** method by the installer) is implementation dependent ([JCRE21]§10.2).

Application note: Other rules governing the installation of an applet, that is, its registration to make it SELECTable by giving it a unique AID, are also implementation dependent (see, for example, [JCRE21], §10).

FMT_SMR.1 SECURITY ROLES

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

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

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

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 [JCRE21] §10.1.4.*

FPT_RCV.3 AUTOMATED RECOVERY WITHOUT UNDUE LOSS

FPT_RCV.3.1/Installer When automated recovery from the failure or service discontinuity reported in Table 13 is not possible, the TSF shall enter a maintenance mode where the ability to return to a secure state is provided.

List of Failure and Services Discontinuity for the maintenance mode
NONE;

Application note: Table 13 – List of Failure and Services to get the maintenance mode stateThe authomated recovery from a failure or service discontinuity is always possible.

Application note: This element is not within the scope of the Java Card specification, which only mandates the behavior of the Java Card System in good working order. Further details on the "maintenance mode" shall be provided in specific implementations. The following is an excerpt from [CC1]:

> In this maintenance mode normal operation might be impossible or severely restricted, as otherwise insecure situations might occur. Typically, only authorized users should be allowed access to this mode but the real details of who can access this mode is a function of class FMT Security management. If FMT does not put any controls on who can access this mode, then it may be acceptable to allow any user to restore the system if the TOE enters such a state. However, in practice, this is probably not desirable as the user restoring the system has an opportunity to configure the TOE in such a way as to violate the TSP.

FPT_RCV.3.2/Installer For ***package loading, applet installation or package/applet deletion failure,*** the TSF shall ensure the return of the TOE to a secure state using automated procedures.

Application note: Should the installer fail during loading/installation of a package/applet, it has to revert to a "consistent and secure state". The JCRE has some clean up duties as well; see [JCRE21], §10.1.4 for possible scenarios. Precise behavior is left to implementers.

Other events such as the unexpected tearing of the card, power loss, and so on. are partially handled by the underlying hardware platform (see the SCPG Security Functional Requirements) and, from the TOE's side, by events "that clear transient objects" and transactional features. See ***FPT_FLS.1.1/JCS***, ***FDP_RIP.1.1/TRANSIENT***, ***FDP_RIP.1.1/ABORTFDP_ROL.1***.

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 ***the transaction capability*** for loss of TSF data or objects within the TSC.

Application note: The quantification is implementation dependent, but some facts can be recalled here. First, the SCP ensures the atomicity of updates for fields and objects (see the SCPG Security Functional Requirements group), and a power-failure during a transaction or the normal runtime does not create the loss of otherwise-permanent data, in the sense that memory on a smart card is essentially persistent with this respect (EEPROM). Data stored on the RAM and subject to such failure is intended to have a limited lifetime anyway (runtime data on the stack, transient objects' contents). According to this, the loss of data within the TSC should be limited to the same restrictions of the transaction mechanism.

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

FRU_RSA.1 MAXIMUM QUOTAS

FRU_RSA.1.1/Installer The TSF shall enforce maximum quotas of the following resources: **imported packages** and **declared classes**, **methods and fields** that **packages** can use **simultaneously**.

Application note: A package may import at most 128 packages and declare at most 255 classes and interfaces. A class can implement a maximum of 128 public or protected instance methods, and a maximum of 128 instance methods with package visibility. These limits include inherited methods. A class instance can contain a maximum of 255 fields, where an **int** data type is counted as occupying two fields ([JCVM21], §2.2.4.2).

6.1.3 ADELG Security Functional Requirements

This group bulks the SFRs related to the deletion of applets and/or packages, enforcing the applet deletion manager **(ADEL) policy** on security aspects outside the runtime. The idea here is that deletion is a critical phase and therefore requires specific treatment.

6.1.3.1 Applet Deletion Manager Policy

FDP_ACC.2: COMPLETE ACCESS CONTROL

FDP_ACC.2.1/ADEL The TSF shall enforce the ***ADEL access control SFP*** on ***S.ADEL, O.JAVAOBJECT, O.APPLET and O.CODE_PKG*** and all operations among subjects and objects covered by the SFP.

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

S.ADEL	The applet deletion manager. It may be an applet ([JCRE22], §11), but its role asks anyway for a specific treatment from the security viewpoint. *This subject is unique.*
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.JAVAOBJECT	Java class instance or array.

Operations (prefixed with "*OP*") of this policy are described in the following table.

Operation	Description
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.

Table 14 – Operation among subject and object of the ADELG SFP

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

FDP_ACF.1.1/ADEL The TSF shall enforce the ***ADEL access control SFP*** to objects based on the following: ***(1) the security attributes of the covered subjects and objects, (2) the list of AIDs of the applet instances registered on the card, (3) the attribute ResidentPackages, which journals the list of AIDs of the packages already loaded on the card, and (4) the attribute ActiveApplets, which is a list of the active applets' AIDs.***

The following table presents some of the security attributes associated to the subjects/objects under control of the policy. However, they are mostly implementation independent.

Subject/Object	Attributes
O.CODE_PKG	package's AID, dependent packages' AIDs, Static References
O.APPLET	Selection state
O.JAVAOBJECT	Owner, Remote

Table 15 – Security attributes of the ADELG SFP

The package's AID identifies the package defined in the CAP file.

When an export file is used during preparation of a CAP file, the version numbers and AIDs indicated in the export file are recorded in the CAP files ([JCVM21], §4.5.2): the dependent packages AIDs attribute allows the retrieval of those identifications.

Static fields of a package may contain references to objects. The Static References attribute records those references.

An applet instance can be in two different selection states: selected or deselected. If the applet is selected (in some logical channel), then in turn it could either be *currently selected* or just *active*. At any time there could be up to four active applet instances, but only one currently selected. This latter is the one that is processing the current command ([JCRE22], §4).

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).

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`**.

Finally, there are needed security attributes that are not attached to any object or subject of the TSP: (1) the ResidentPackages Versions (or Resident Image,[JCVM21],§4.5) and AIDs. They describe the packages that are already on the card, (2) the list of registered applet instances and (3) the ActiveApplets security attribute. They are all attributes internal to the VM,

that is, not attached to any specific object or subject of the SPM. These attributes are TSF data that play a role in the SPM.

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 by the ADEL SFP:***

The subjects of this policy is ***S.ADEL***.

Some basic common specifications are required in order to allow Java Card applets and packages to be deleted without knowing the implementation details of a particular deletion manager. In particular, this policy introduces a notion of **reachability**, which provides a general means to describe objects that are referenced from a certain applet instance or package.

In the context of this policy, an object O is reachable if and only if either: (1) the owner of O is a registered applet instance A (O is reachable from A), (2) a static field of a loaded 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), and (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 ([JCRE22], §11.3.4.1, **Applet Instance Deletion**). The ***S.ADEL*** may perform ***OP.DELETE_APPLET*** upon an ***O.APPLET*** only if, (1) ***S.ADEL*** is currently selected, (2) ***O.APPLET*** is deselected 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 ([JCRE22], §8.5) ***O.JAVAOBJECT*** is remote reachable.

R.JAVA.15 ([JCRE22],§11.3.4.1, **Multiple Applet Instance Deletion**). The ***S.ADEL*** may perform ***OP.DELETE_APPLET*** upon several ***O.APPLET*** only if, (1) ***S.ADEL*** is currently selected, (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 ([JCRE22], §8.5) ***O.JAVAOBJECT*** is remote reachable.

R.JAVA.16 ([JCRE22], §11.3.4.2, **Applet/Library Package Deletion**). The ***S.ADEL*** may perform ***OP.DELETE_PCKG*** upon an ***O.CODE_PCKG*** only if, (1) ***S.ADEL*** is currently selected, (2) no reachable ***O.JAVAOBJECT***, from a package distinct from ***O.CODE_PCKG*** that is an instance of a class that belongs to

O.CODE_PCKG exists on the card and (3) there is no package loaded on the card that depends on *O.CODE_PCKG.*

R.JAVA.17 ([JCRE22], §11.3.4.3, Applet **Package and Contained Instances Deletion**). The *S.ADEL* may perform *OP.DELETE_PCKG_APPLET* upon an *O.CODE_PCKG* only if, (1) *S.ADEL* is currently selected, (2) no reachable *O.JAVAOBJECT*, from a package distinct from *O.CODE_PCKG*, which is an instance of a class that belongs to *O.CODE_PCKG* exists on the card, (3) there is no package loaded on the card that depends on *O.CODE_PCKG* and (4) for every *O.APPLET* of those being deleted it holds that: (i) *O.APPLET* is deselected 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 ([JCRE22],§8.5) *O.JAVAOBJECT* is remote reachable.

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

Application note: However, the *S.ADEL* may be granted privileges ([JCRE22], §11.3.5) to bypass the preceding policies. For instance, the logical deletion of an applet renders it un-selectable; this has implications on the management of the associated TSF data (see application note of **FMT_MTD.1.1/JCRE**).

FDP_ACF.1.4/ADEL The TSF shall explicitly deny access of ***any subject but the S.ADEL to O.CODE_PKG or O.APPLET for the purpose of deleting it from the card.***

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/ADEL The TSF shall enforce the ***ADEL access control SFP*** to restrict the ability to **modify the ActiveApplets** security attribute to **the JCRE (*S.JCRE*).**

Application note: The modification of the ActiveApplets security attribute should be performed in accordance with the rules given in [JCRE22], §4.

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/ADEL The TSF shall be capable of performing the following security management functions: ***modification of the ActiveApplets security attributes***.

Application note: Added because [CCFI_065]

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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.

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

FMT_SMR.1 SECURITY ROLES

FMT_SMR.1.1/ADEL The TSF shall maintain the roles: *the* ***applet deletion manager***.

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

6.1.3.2 Additional Deletion Requirements

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

FDP_RIP.1.1/ADEL The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation 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.***

Application note: Deleted freed resources (both code and data) may be reused, depending on the way they were deleted (logically or physically). Requirements on *de-allocation* during applet/package deletion are described in [JCRE22], §11.3.4.1, §11.3.4.2 and §11.3.4.3.

Application note: There is no conflict with **FDP_ROL.1** requirements appearing in the document as of the bounds on the rollback: the deletion operation is out of the scope of the rollback (***FDP_ROL.1.1/FIREWALL***, p.58).

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

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* [JCRE22], §11.3.4.

Application note:

6.1.4 RMIG Security Functional Requirements

This group is mainly devoted to specifying the policies that control the access to remote objects and the flow of information that takes place when the RMI service is used. There are specific control rules concerning the access to remote objects. The rules relate mainly to the lifetime of their corresponding 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 are required to be allocated on the card as global arrays, the storage of references to those arrays must then be restricted as well.

6.1.4.1 JCRMI Policy

The JCRMI policy embodies both an access control and an information flow control policy.

FDP_ACC.2: COMPLETE ACCESS CONTROL

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

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

`S.CAD`	The CAD. In the scope of this policy it represents the actor that requests, by issuing commands to the card, for RMI services.
`S.JCRE`	The JCRE is responsible on behalf of the card issuer of the bytecode execution and runtime environment functionalities. In the context of this security policy, the JCRE is in charge of the execution of the commands provided to (1) obtain the initial remote reference of an applet instance and (2) perform Remote Method Invocation.
`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`** ([JCAPI22]).
`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.

Operations (prefixed with "*OP*") of this policy are described in the following table.

Operation	Description
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.

Table 16 - Operations among subjects and objects covered by the of the RMIG SFP

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

FDP_ACF.1.1/JCRMI The TSF shall enforce the ***JCRMI access control SFP*** to objects based on the following: ***(1) the security attributes of the covered subjects and objects, (2) the list of AIDs of the applet instances registered on the card and (3) the attribute ActiveApplets, which is a list of the active applets' AIDs.***

The following table presents the security attributes associated to the objects under control of the policy.

Object	Attributes
O.APPLET	Package's AID or *none*
O.REMOTE_OBJ	Owner, class, Identifier, Exported
O.REMOTE_MTHD	Identifier
O.RMI_SERVICE	Owner, Returned References

Table 17 – Security attributes of the RMIG SFP

The package's AID identifies the package defined in the CAP file.

An applet instance can be in two different selection states: selected or deselected. If the applet is selected (in some logical channel), then in turn it

could either be *currently selected* or just *active*. At any time there could be up to four active applet instances, but only one currently selected. This latter is the one that is processing the current command ([JCRE22], §4).

The **owner** of a remote object is the applet instance that created the object. The **class** attribute identifies the implementation class of the remote object. The **remote object Identifier** is a number that uniquely identifies a remote object. The attribute **Exported** indicates whether the remote object is exportable or not.

A **remote method Identifier** is a number that uniquely identifies a remote method within a certain remote class.

The **owner** of an ***O.RMI_SERVICE*** is the applet instance that created the object. The attribute **Returned References** lists the remote object references that have been sent to the CAD during the applet selection session. This attribute is implementation dependent.

Finally, there are some security attributes that are not attached to any object or subject of the TSP: (1) the list of registered applet instances and (2) the ActiveApplets security attribute. They are all attributes internal to the VM that is, not attached to any specific object or subject of the SPM. These attributes are TSF data that play a role in the SPM.

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 by the ***JCRMI SFP***:

R.JAVA.18 The ***S.CAD*** may perform ***OP.GET_ROR*** upon an ***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 The ***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 [JCRE22], §8.5) and belongs to the value of the attribute 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_OBJECT*** to which ***O.ROR*** makes reference.

Application note: The validity of a remote object reference is specified as a lifetime characterization. The security attributes involved in the rules that determine what a valid remote object reference is are the attribute Returned References of the ***O.RMI_SERVICE*** and the attribute ActiveApplets (see ***FMT_REV.1.1/JCRMI*** and ***FMT_REV.1.2/JCRMI***).

Application note: The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([JCRE22], §8.5.2 and [JCAPI22]).

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

FDP_ACF.1.4/JCRMI 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.

FDP_IFC.1 SUBSET INFORMATION FLOW CONTROL

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

Subjects[7] (prefixed with an "`S`") and information (prefixed with an "`I`") covered by this policy are:

Subject/Information	Description
`S.JCRE`	As in the Access control policy
`S.CAD`	As in the Access control policy
`I.RORD`	Remote object reference descriptors

Table 18 – Subject and Information of the RMIG SFP

A remote object reference descriptor provides 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.

Application note: Array parameters of remote method invocations must be allocated on the card as global arrays objects. References to global arrays cannot be stored in `class` variables, instance variables or array components. The control of the flow of that kind of information has already been specified in ***FDP_IFC.1.1/JCVM.***

There is a unique operation in this policy:

Operation	Description
`OP.RET_RORD(S.JCRE,S.CAD,I.RORD)`	Send a remote object reference descriptor to the CAD.

Table 19 – Operation of the RMIG SFP

A remote object reference descriptor is sent from the card to the CAD either as the result of a successful applet selection command ([JCRE22], §8.4.1), and in this case it describes, if any, the initial remote object

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

reference of the selected applet; or as the result of a remote method invocation ([JCRE22],§8.3.5.1) .

FDP_IFF.1 SIMPLE SECURITY ATTRIBUTES

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: ***the*** *security attribute Exported of the information.*

The following table summarizes which security attribute is attributed to which subject/information.

Subject/Information	Attributes
`S.JCRE`	None
`S.CAD`	None
`I.RORD`	ExportedInfo (Boolean value)

Table 20 – Security attributes of the RMIG SFP

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

FDP_IFF.1.2/JCRMI The TSF shall permit an information flow between a controlled subject and controlled information through a controlled operation if the following rule holds:

An operation `OP.RET_RORD(S.JCRE,S.CAD,I.RORD)` is permitted only if the attribute ExportedInfo `I.RORD` has the value **"true"** ([JCRE22], §8.5).

FDP_IFF.1.3/JCRMI The TSF shall enforce ***the following additional information flow control SFP rules: none***

FDP_IFF.1.4/JCRMI The TSF shall provide ***the following additional SFP capabilities: none***.

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/JCRMI The TSF shall enforce the ***FIREWALL access control SFP and the JCVM information flow control SFP*** to restrict the ability to **modify the ActiveApplets** security attribute to **the JCRE (*S.JCRE*).**

Application note: The modification of the ActiveApplets security attribute should be performed in accordance with the rules given in [JCRE22], §4.

FMT_MSA.1.1/EXPORT The TSF shall enforce the ***JCRMI access control SFP and the JCRMI information flow control SFP*** to restrict the ability to **modify the security attribute Exported of an O.REMOTE_OBJ to its owner.**

Application note: The Exported status of a remote object can be modified by invoking its methods export() and unexport(), and only the owner of the object may perform the invocation without raising a SecurityException (javacard.framework.service.CardRemoteObject). However, even if the owner of the object may provoke the change of the security attribute value, the modification itself could be performed by the JCRE.

FMT_MSA.1.1/REM_REFS The TSF shall enforce the ***JCRMI access control SFP and the JCRMI information flow control SFP*** to restrict the ability to **modify the security attribute Returned References of an O.RMI_SERVICE to its owner.**

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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.

Application note: 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.

Application note: There is one default value for the *SELECTed applet context* that is the *default applet identifier's context*, and one default value for the *active context*, that is "*JCRE*".

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

Application note: The intent is to have none of the identified roles to have privileges with regards to the default values of the security attributes. Notice that creation of objects is an operation controlled by the ***FIREWALL SFP***; the latitude on the parameters of this operation is described there.

FMT_REV.1 REVOCATION

FMT_REV.1.1/JCRMI The TSF shall restrict the ability to revoke the **Returned References security attribute of an O.RMI_SERVICE** to **the JCRE**.

FMT_REV.1.2/JCRMI The TSF shall enforce the rules ***that determine the lifetime of remote object references***.

Application note: The rules previously mentioned are described in [JCRE22], §8.5.

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/JCRMI The TSF shall be capable of performing the following security management functions: ***modification of the ActiveApplets security attributes, modification of Exported of an O.REMOTE_OBJ security attribute, modification of the secutiy attribute Returned References of an O.RMI_SERVICE***

Application note: Added because [CCFI_065]

FMT_SMR.1 SECURITY ROLES

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

Application note: applets own Remote interface objects and may choose to allow or forbid their exportation, which is managed through a security attribute.

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

6.1.5 LCG Security Functional Requirements

The security issues introduced by logical channels are mainly related to the access to SIO objects owned by legacy applets as well as to the clearing of transient data which is shared by applet instances which are concurrently active in different logical channels. Accordingly, this group introduces a reformulation of the ***FIREWALL SFP*** (*FDP_ACF.1.1/FIREWALL, FDP_ACF.1.2/FIREWALL, FMT_MSA.1.1/JCRE)* specified in the group CoreG Security Functional Requirements and a modification to a component of the security requirement for residual information protection (*FDP_RIP1.1/TRANSIET*).

6.1.5.1 Firewall Policy

See 6.1.1.1 Firewall Policy in the CoreG Security Functional Requirements.

6.1.5.2 Additional Requirements on Logical Channels

See ***FDP_RIP.1.1/TRANSIENT*** in the pharagraph 6.1.1.2 Application Programming Interface ***FDP_RIP.1.1/TRANSIENT, FDP_RIP.1.1/TRANSIENT***.

6.1.6 ODELG Security Functional Requirements

The following requirements are concerned with the secure deletion of information provoked by the object deletion mechanism. This mechanism is triggered by the applet who owns the deleted objects by invoking a specific API method.

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

FDP_RIP.1.1/ODEL The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation 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().`**

Application note: Freed data resources resulting from the invocation of the method **`javacard.framework.JCSystem.requestObjectDeletion()`** may be reused. Requirements on *de-allocation* after the invocation of the method are described in [JCAPI22].

Application note: There is no conflict with **FDP_ROL.1** here because of the bounds on the rollback mechanism: the execution of **`requestObjectDeletion()`** is not in the scope of the rollback because it must be performed in between APDU command processing, and therefore no transaction can be in progress.

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

FPT_FLS.1.1/ODEL The TSF shall preserve a secure state when the following type of failure occurs: *the object deletion functions fail to delete all the unreferenced objects owned by the applet that requested the execution of the method*

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

6.1.7 CarG Security Functional Requirements

This group of requirements applies to those configurations where the bytecode verifier is not embedded on the card. The TOE shall include requirements for preventing the installation of a package that has not been bytecode verified, or that has been modified after bytecode verification.

FCO_NRO.2 ENFORCED PROOF OF ORIGIN

FCO_NRO.2.1/CM The TSF shall enforce the generation of evidence of origin for transmitted application packages at all times.

Application note: If this is the case and a new application package is received by the card for installation, the card manager shall first check that it actually comes from the verification authority . The verification authority is the entity responsible for bytecode verification.

FCO_NRO.2.2/CM 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.

FCO_NRO.2.3/CM The TSF shall provide a capability to verify the evidence of origin of information to ***the recipient*** given the following limitations: the card manager performs an immediate verification of the origin of the package using an electronic signature mechanism, and no evidence is kept on the card for future verifications

Application note: The exact limitations on the evidence of origin are implementation dependent. In most of the implementations, the card manager performs an immediate verification of the origin of the package using an electronic signature mechanism, and no evidence is kept on the card for future verifications.

FIA_UID.1 TIMING OF IDENTIFICATION

FIA_UID.1.1/CM The TSF shall allow the following TSF-mediated actions on behalf of the user to be performed before the user is identified: none.

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.

Application note: The list of TSF-mediated actions is implementation-dependent, but package installation requires the user to be identified. Here by user is meant the one(s) that in the Security Target shall be associated to the role(s) defined in the component *FMT_SMR.1/CM*.

FDP_IFC.2 COMPLETE INFORMATION FLOW CONTROL

FDP_IFC.2.1/CM The TSF shall enforce the ***PACKAGE LOADING information flow control SFP*** on ***S.CRD, S.BCV, S.SPY*** and all operations that cause that information to flow to and from subjects covered by the SFP.

Subjects (prefixed with an "*S*") covered by this policy are those involved in the reception of an application package by the card through a potentially unsafe communication channel:

Subject	Description
S.BCV	The subject representing who is in charge of the bytecode verification of the packages (also known as the verification authority).
S.CRD	The on-card entity in charge of package downloading.
S.SPY	Any other subject that may potentially intercept, modify, or permute the messages exchanged between the former two subjects.

Table 21 – Subject and object of the CarG SFP

The operations (prefixed with "*OP*") 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 the attacker. Moreover, the attacker may capture any message sent through the communication channel and send its own messages to the other subjects.

Operation	Description
OP.SEND(M)	A subject sends a message M through the communication channel.
OP.RECEIVE(M)	A subject receives a message M from the communication channel.

Table 22 – Operation of the CarG SFP

The information (prefixed with an "**I**") controlled by the typing 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 (either ***S.BCV*** or ***S.SPY)*** in the communication protocol.

Information	Description
I.APDU	Any APDU sent to or from the card through the communication channel.

Table 23 –Information of the CarG SFP

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

FDP_IFF.1 SIMPLE SECURITY ATTRIBUTES

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:

Subject/Object	Attributes
`S.BCV`	Keys used to calculate the MAC of messages to be sent
`S.CRD`	Keys used to verify the MAC of received messages, X-MAC
`S.SPY`	None
`I.APDU`	Block sequence counter, C-MAC

Table 24 - Security Attribute of the PACKAGE LOADING information flow control SFP

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

Name	Description
Key	Keys used to calculate/verify the MAC of messages
Block sequence counter	Ordinal of each piece in the decomposition of the package
C-MAC	MAC calculated by the S.BCV
X-MAC	MAC calculated by `S.CRD`

Table 25 – Security attributes values

Application note: Security Attribute values of the PACKAGE LOADING information flow control SFPThe security attributes used to enforce the PACKAGE LOADING SFP are implementation dependent. More precisely, they depend on the communication protocol enforced between the CAD and the card. For instance, some of the attributes that can be used are : (1) the keys used by the subjects to encrypt/decrypt their messages; (2) the number of pieces the application package has been split into in order to be sent to the card; (3) the ordinal of each piece in the decomposition of the package, and so on. See Appendix E of [GP].

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

R.GP.1 the subject `S.BCV` shall perform `OP.SEND` upon any application package. The application package is splitted in one or more `I.APDU`. For each `I.APDU`, `S.BCV` evaluates C-MAC

using the relevant Key. ***S.BCV*** put the C-MAC and the Sequence Block Counter in ***I.APDU.***

R.GP.2 the subject ***S.CRD*** shall perform ***OP.RECEIVE*** and accept an ***I.APDU*** if and only if it comes from the subject S.BCV and it has received without modification and in the right order (X-MAC calculated using the relevant Key is equal to the C-MAC).

Application note: The precise set of rules to be enforced by the function is implementation dependent. The whole exchange of messages shall verify at least the following two rules: (1) the subject S.CRD shall accept a message only if it comes from the subject S.CAD; (2) the subject S.CRD shall accept an application package only if it has received without modification and in the right order all the APDUs sent by the subject S.CAD.

.Note: The communication protocol used is the GP SCP02 specified in Appendix E of [GP].

FDP_IFF.1.3/CM The TSF shall enforce the ***following additional information flow SFP rules: none***.

FDP_IFF.1.4/CM The TSF shall provide ***the following list of additional SFP capabilities: none.***

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

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

FDP_UIT.1 DATA EXCHANGE INTEGRITY

The bytecode verifier can be seen as an external IT product, and packages to be loaded on the card are user data in transit from that external product to the Java Card System.

FDP_UIT.1.1/CM The TSF shall enforce the ***PACKAGE LOADING information flow control*** SFP to be able to ***receive*** user data in a manner protected from ***modification, deletion, insertion and replay*** errors***.***

Application note: Modification errors should be understood as modification, substitution, unrecoverable ordering change of data and any other integrity error that may cause the application package to be installed on the card to be different from the one sent by the CAD.

FDP_UIT.1.2/CM 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.

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/CM The TSF shall enforce the ***PACKAGE LOADING information flow control SFP*** to restrict the ability to modify the security attributes ***Key, Block Sequence Counter and C-MAC to S.BCV***.

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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 *SF*P.

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/CM The TSF shall be capable of performing the following security management functions: ***modification of the security attributes Key, Block Sequence Counter and C-MAC to S.BCV.***

Application note: Added because [CCFI_065].

FMT_SMR.1 SECURITY ROLES

FMT_SMR.1.1/CM The TSF shall maintain the roles: the S.BCV.

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

FTP_ITC.1 INTER-TSF TRUSTED CHANNEL

The CAD can be seen as a remote IT product, and packages to be loaded on the card shall be transmitted using an inter-TSF trusted channel to prevent them from being modified during downloading. Such trusted channel connects the embedded Java Card System to the secured environment of the card issuer where the package has been verified.

FTP_ITC.1.1/CM The TSF shall provide 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.

FTP_ITC.1.2/CM The TSF shall permit ***the CAD placed in the card issuer secured environment*** to initiate communication through the trusted channel.

FTP_ITC.1.3/CM The TSF shall initiate communication through the trusted channel for ***installing a new application package on the card.***

Application note: there is no dynamic package loading on the Java Card platform. New packages can be installed on the card only on demand of the card issuer .

6.2 TOE SECURITY ASSURANCE REQUIREMENTS

The assurance requirement is **EAL 4 augmented**.

REQUIREMENT	NAME	TYPE
EAL 4	Methodically designed, tested, and reviewed	Assurance level

The assurance requirements ensure, among others, the security of the TOE during its development and production. We present here some application notes on the assurance requirements included in the EAL 4. *These are not to be considered as iteration or refinement of the original components.*

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

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

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

These SARs ensure that the TOE will be able to meet its security requirements and fulfill its objectives. The Java Card System shall implement the [JCAPI22]. The implementation of the Java Card API shall be designed in a secure manner, including specific techniques to render sensitive operations resistant to state-of-art attacks.

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

This SAR ensures the integrity of the TOE and its documentation during the transfer of the TOE between all the actors appearing in the first two stages. Procedures shall ensure protection of TOE

material/information under delivery and storage that corrective actions are taken in case of improper operation in the delivery process and storage and that people dealing with the procedure for delivery have the required skills.

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

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

The applet and API packages programmers should have a complete understanding of the concepts defined in [JCRE22] and [JCVM22]. They must delegate key management, PIN management and cryptographic operations to dedicated APIs. They should carefully consider the effect of any possible exception or specific event and take appropriate measures (such as catch the exception, abort the current transaction, and so on.). They must comply with all the recommendations given in the platform programming guide as well. Failure to do so may jeopardize parts of (or even the whole) applet and its confidential data.

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

- ***ALC_DVS.1*** Identification of security measures
- ***ALC_LCD.1*** Developer defined life-cycle model
- ***ALC_TAT.1*** Well-defined development tools

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

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

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

- ***AVA_MSU.2*** Validation of analysis

This SAR ensures that the guidance on installation, generation, and start-up procedures is not misleading, unreasonable or conflicting, whether secure procedures for all modes of operation have been addressed, and whether use of the guidance will facilitate prevention and detection of insecure TOE states.

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

The objectives of this SAR are to determine whether SOF claims are made in the ST for all non-cryptographic, probabilistic or permutational mechanisms and whether the developer's SOF claims made in the ST are supported by an analysis that is correct.

The strength of function applied in AVA_SOF.1 is SOF High.

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

- ***AVA_VLA.3*** Moderately Resistant

EAL4 requires vulnerability assessment through imposition of ***AVA_VLA.2***. This dictates a review of identified vulnerabilities only. The component ***AVA_VLA.3*** requires that a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***.

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

EAL4 requires through imposition of ***ADV_IMP.1*** the description of a subset of the implementation representation in order to capture the detailed internal working of the TSF. The component ***ADV_IMP.2*** requires the developer to provide the implementation representation for the entire TSF.
Table 26 reports a summary of the assurance requirements. In gray are highligthed the component for the augumentation.
The developer will address the TOE security assurance requirement by producing dedicated documents. The documents to be delivered cover each one TOE security assurance requirement.

Assurance Class	Assurance Component	Description
Configuration Management	ACM_AUT.1	Partial CM Automation
	ACM_CAP.4	General Support and Acceptance Procedures
	ACM_SCP.2	Development tools CM Coverage
Delivery and Operation	ADO_DEL.2	Detection of Modification
	ADO_IGS.1	Installation, generation and start up procedures
Development	ADV_FSP.2	Functional Specification
	ADV_HLD.2	High Level Design
	ADV_IMP.2	Implementation of the TSF
	ADV_LLD.1	Descriptive Low Level Design
	ADV_RCR.1	Informal correspondence demonstration
	ADV_SPM.1	Informal TOE security policy model
Guidance Documents	AGD_ADM.1	Administrator Guidance
	AGD_USR.1	User Guidance
Life cycle support	ALC_DVS.1	Identification of security measures
	ALC_LCD.1	Developer defined life-cycle model
	ALC_TAT.1	Well-defined development tools
Tests	ATE_COV.2	Analysis of coverage
	ATE_DPT.1	Testing: high-level design
	ATE_FUN.1	Functional testing
	ATE_IND.2	Independent testing - sample
Vulnerability assessment	AVA_MSU.2	Validation of Analysis
	AVA_SOF.1	Strength of function analysis
	AVA_VLA.3	Vulnerability analysis

Table 26 – Assurance Requirements Summary

6.3 SECURITY REQUIREMENTS FOR IT ENVIRONMENT

6.3.1 SCPG Security Functional Requirements

This group contains the security requirements for the smart card platform, that is, operating system and chip that the Java Card System is implemented upon. It does not define requirements for the TOE but for its IT environment. The requirements are expressed in terms of security functional requirements from [CC2].

UNDERLYING ABSTRACT MACHINE TEST (FPT_AMT)

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

Application note: The abstract machine that underlies the TSF comprises the lower levels of the SCP, that is, the OS and its dedicated native applications and/or APIs (for instance, hardware cryptographic functions/buffers), as well as the IC. Self-test of these components is, as an example, included in [PP0010]. These tests are initiated by the TSF of the SCP itself.

FAIL SECURE (FPT_FLS)

FPT_FLS.1.1/SCP The TSF shall preserve a secure state when the following types of failures occur:

- card lifecycle corruption,
- EEPROM Writing Errors,
- Voltage or frequency out of range values,
- unexpected abortion of a TSF due to an external event.

FAULT TOLERANCE (FRU_FLT)

FRU_FLT.1.1/SCP The TSF shall ensure the operation of maintaining EEPROM consistencewhen the following failures occur:

- Unexpected power loss

TSF PHYSICAL PROTECTION (FPT_PHP)

FPT_PHP.3.1/SCP The TSF shall resist ***to the following list of tampering scenarios*** to the ***following list of TSF devices/elements*** by responding automatically such that the TSP is not violated.

Elements	Physical tampering Scenarios
Card Life cycle state	Erasure of the card life cycle state by means of an hardware tampering
Clock	Reduction of clock frequency to stop the TOE during specific operation
Clock	Increase the clock frequency to corrupt TOE operation behavior
Temperature	Use the TOE in out of range temperature
Voltage Supply	Voltage supply out of range to corrupt the TOE operation behavior

Table 27 – Elements and Physical Tamper Scenario

DOMAIN SEPARATION (FPT_SEP)

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

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

Application note: The use of "security domain" here refers to execution space, and should not be confused with other meanings of security domains.

REFERENCE MEDIATION (FPT_RVM)

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

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

The TSF and TSC stated in these three components refer to that of the SCP.

TRUSTED RECOVERY (FPT_RCV)

FPT_RCV.3.1/SCP When automated recovery from the list of failure or service discontinuity reported in Table 13 is not possible, the TSF shall enter a maintenance mode where the ability to return to a secure state is provided.

List of Failure and Services Discontinuity for the maintenance mode
NONE;

Table 28 – List of Failure and Services to get the maintenance mode state

FPT_RCV.3.2/SCP For unespected power loss, voltage or frequency out of range, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

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

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

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

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

6.3.2 *CMGRG* Security Functional Requirements

This group contains the security requirements for the card manager. These are requirements for the IT environment of the TOE. They are all expressed in terms of security functional requirements from [CC2].

The security requirements below helps define a policy for controlling access to card content management operations and for expressing card issuer security concerns. This policy shall be highly dependent on the particular security and card management architecture present in the card.

FDP_ACC.1 SUBSET ACCESS CONTROL

FDP_ACC.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** on **S.ADMINISTRATOR, S.CARD_MGR, O.PACKAGE, O.APPLET**.

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

Subject/Object	Description
S.ADMINISTRATOR	*The card administrator*
S.CARD_MGR	The card manager
O.PACKAGE	A package
O.APPLET	An applet

Table 29 – Subject and object of the CMGRG group

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

Operation	Description
OP.AUTHENTICATION(S.CARD_MGR, authToken*)*	Authentication of the *S.ADMINISTRATOR* by the *S.CARD_MGR.*
OP.LOAD(O.PACKAGE, pkAID*)*	Load a package with the assigned pkAID
OP.INSTALL(O.APPLET, pkAID*,* appAID*,* instAID*)*	Install an O.APPLET of the Applet class with appAID from the O.PACKAGE with pkAID, assigning instAID.
OP.DELETE_APPLET(O.APPLET , instAID*)*	Delete the O.APPLET with instAID.
OP.DELETE_PCKG(O.PACKAGE, pkAID*)*	Delete the O.PACKAGE with pkAID.
OP.DELETE_PCKG_APPLET(O.PACKAGE, pkAID*)*	Delete the O.PACKAGE with pkAID and all the O.APPLET installed by this package.
OP.CARD_CONTENT_MGT(…)	Any operation of this SFP different from *OP.AUTHENTICATION()*

Table 30 – Operation among subject and object of the CMGRG SFP

Application note: It should be noticed that TSF here refers to the security functions of the environment, rather than security functions of the TOE.

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

FDP_ACF.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to objects based on ***(1) administrator authentication status.***

The following table relates the security attribute to the subject/object of the policy.

Subject/Object	*Attributes*
`S.ADMINISTRATOR`	None
`S.CARD_MGR`	Authentication Status
`O.PACKAGE`	None
`O.APPLET`	None

Table 31 – Security attribute of the CMGRG SFP

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

Attributes	**Value**
Authentication Status	TRUE *or* FALSE

Table 32 - Values of the security attribute

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

R.CM 1 **- An S.ADMINISTRATOR is enabled to perform an `OP.CARD_CONTENT_MGT()` if and only if the authentication status of the S.CMGR is true.**

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to restrict the ability to ***modify*** the security attributes ***authentication status*** to S.CARD_MGR.

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/CMGR The TSF shall be capable of performing the following security management functions: ***modification of the authentication status***.

Application note: Added because [CCFI_065].

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

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

FMT_SMR.1 SECURITY ROLES

FMT_SMR.1.1/CMGR The TSF shall maintain the roles: ***S.CARD_MGR***.

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

FIA_UID.1 TIMING OF IDENTIFICATION

FIA_UID.1.1/CMGR The TSF shall allow ***the TSF mediated following action*** on behalf of the user to be performed before the user is identified: ***none***.

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

6.3.3 BCVG Security Functional Requirements

This group of requirements concerns bytecode verification. A bytecode verifier can be understood as a process that acts as a filter on a CAP file verifying that the bytecodes of the methods defined in the file conform to certain well-formed requirements. The solution described in [JCBV], and adopted by ST-Incar, is based on a data flow analysis and makes use of an abstract interpreter. The abstract interpreter simulates execution of each instruction, using types of the data being operated on instead of values. For each instruction, the state of the operand stack and local variables are compared to the type(s) required during execution, and then are updated according to the operation of the instruction. The main component of this group of functional requirements is an information flow control policy, which describes the constraints imposed on the operations (the bytecodes) that make information flow between the subjects (local variables, operand stack, fields).

The group is composed of three sub-groups. The first one constitutes a complete information flow control policy with hierarchical attributes, which describes the type constraints imposed on the bytecodes. That typing policy strongly depends on having a secure configuration of the attributes it is based on. Such secure configurations are strongly related to the constraints imposed on the structure of the CAP file format by Sun specifications, and constitute a second important sub-group of requirements. Finally, the third sub-group requires bytecode verification to prevent any operand stack overflow that could arrive during the interpretation of bytecodes.

FDP_IFC.2 COMPLETE INFORMATION FLOW CONTROL

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

Subjects[8] (prefixed with an "*S*") covered by this policy are:

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

Table 33 – Subjects of the BCVG SFP

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

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

Table 34 – Operation of the BCVG SFP

The information (prefixed with an "**I**") controlled by the typing policy are the bytes, shorts, integers, references and return addresses contained in the

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

different storage units of the JCVM (local variables, operand stack, static fields, instance fields and array positions).

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

Table 35 – Information of the BCVG SFP

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

FDP_IFF.2 HIERARCHICAL SECURITY ATTRIBUTES

See ***FMT_MSA.1*** for more information about security attributes.

FDP_IFF.2.1/BCVG The TSF shall enforce the ***TYPING information flow control SFP*** based on the following types of subject and information security attributes: ***(1) type attribute of the information, (2) type attribute of the storage units of the JCVM, (3) class attribute of the fields and methods, (4) bounds attribute of the methods.***

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

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

Table 36 - security attributes defined for the BCVG SFP

The following table describes the security attributes.

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

Table 37 - Security attributes description for the BCVG SFP

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

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

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

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

Table 38 – Values of the security attributes for the BCVG SFP

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

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

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

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

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

R.JAVA.8 If the bytecode modifies a local variable with a value with attribute *TYPE* T, it must be recorded that the local variable now

contains a value of that type. In addition, the variable shall be among the local variables of the method.

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

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

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

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

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

Application note: The rules described above are strongly inspired in the rules described in section 4.9 of [JVM], ***Second Edition***. The complete set of typing rules can be derived from the "Must" clauses from Chapter 7 of [JCVM21] as instances of the rules defined above.

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

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

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

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

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

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

b) There exists a least upper bound in the set of security attributes, such that, given any two valid security attributes, there is a valid

security attribute that is greater than or equal to the two valid security attributes; and

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

Application note: The order relationship between Java Card types is described, for instance, in the description of the **checkcast** bytecode of [JCVM21]. That relation is with the following rules:

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

(See ***FMT_SMR.1.1/BCV*** (p. 101) for the roles)

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

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

Application note: The TYPE attribute of the local variables and the operand stack positions is identified to the attribute of the information they hold. Therefore, this security attribute is possibly modified as information flows. For instance, the rules of the typing function enable information to flow from a local variable *lv* to the operand stack by the operation `sload`, provided that the value of the type attribute of *lv* is ***short***. This operation hence modifies the type attribute of the top of the stack. The modification of the security attributes should be done according to the typing rules derived from ***Chapter 7 of [JCVM21]***.

FMT_MSA.2 SECURE SECURITY ATTRIBUTES

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

Application note: During the type verification of a method, the bytecode verifier makes intensive use of the information provided in the CAP format like the

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

- Correspondences between the different components of the CAP file (for instance, each class in the class component has an entry in the descriptor component).
- Pointer soundness (example: the index argument in a static method invocation always has an entry in the constant pool);
- Absence of hanged pointers (example: each exception handler points to the beginning of some bytecode);
- Redundant information (enabling different ways of searching for it);
- Conformance to the Java Language Specification respecting the access control features mentioned in §2.2 of [JCVM22].
- Packages that are loaded post-issuance can not contain native code.

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

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

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

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/BCV The TSF shall be capable of performing the following security management functions: ***modification of TYPE security attribute of the fields, modification of TYPE security attribute of methods and modification of TYPE security attribute of local variables and operand stack position***.

Application note: Added because [CCFI_065].

FMT_SMR.1 SECURITY ROLES

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

Note: the actual set of roles defined in the ST depends on the configuration.

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

FRU_RSA.1 MAXIMUM QUOTAS

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

7 TOE Summary Specifications

7.1 TOE SECURITY FUNCTIONS

Security Function name	Description
SF_CARD_MNGT	*Card Management*
SF_CRYPTO_KEY	*Cryptographic Key Management*
SF_PIN	*PIN management*
SF_CRYPTO_OP	*Cryptographc Computation*
SF_FIREWALL	*Obect access controller (firewall)*
SF_OBJ_MNGT	*Object Management*
SF_RMI	*Rmeote method invocation*
SF_POST	*Power on Self test*
SF_TRANSACTION	*Transaction Managment*

Table 39 - TOE Security Functions (SF) summary

7.1.1 SECURITY FUNCTION SF_CARD_MNGT

It is responsible for the card content management, applet selection and applet lifetime.

Its main task are:

- to import packages on the card (loading) enforcing the importation rules;
- to install applet by calling its install() method;
- to manage the application lifetime calling the methods `install()`, `select()`, `deselect()`, `process()`, `uninstall()`, according [JCRE22];
- to delete applets and objects according the [JCRE22];
- to enforce the integrity of the package and applet registries;
- to manage the secure channel protocol (SCP 02 [GP]);

SCP 02 includes:

- a mutual authentication procedure to authentication of the actors (the TOE and the administrator) before performing card management,
- integrity on the received data during the loading sessions,
- and data origin authentication on the received dataduring the loading sessions.

The mutual authentication procedure is based on a challenge response protocol. The off card entity equipment (the equipment of the administrators) send an "host" challenge (random data) to the card. The card generates its own "card" challenge, generates session key and generates a cryptographic value (Card Cryptogram) based on the "host" challenge and on the session key; the card send the card Cryptogram and the card challenge to the off card entity. The off card entity calculates the session key, recalculates the Card Cryptogram and authenticate the card by comparing the "card" cryptogram returned by the card with its own calculated "card" cryptogram. Moreover the off card entity generates a cryptographic value (Host Cryptogram) using the session key and the card challenge; the off card entity then send back this value to the card. The card authenticate the off card entity and hence the administrator by means of the previously described procedure. The cryptographic algorithm used in the authentication protocol is based on 3-DES.

One permutational mechanisms has been found in this SF: the mutual authentication procedure;

The strength of this function is SOF-High.

7.1.2 SECURITY FUNCTION SF_CRYPTO_KEY

This security function is related to all the operations on the application keys.

In particular SF_CRYPTO_KEY:

- manages the key distribution;
- manages the access to keys;
- manages the key destruction;
- menages the keys generation; allowed key length for RSA algorithm are 512, 768, 1024 bit;.

Moreover the security function assures the integrity of the stored keys.

The security function relies on a cryptolibrary provided by the chip manufacturer to perform the RSA keys generation. The cryptolibrary uses an hardware random number generator in order to generate prime numbers and the keys. Both cryptolibrary and the hardware random number generator are not part of the TOE.

This function has no SOF claims.

7.1.3 SECURITY FUNCTION SF_CRYPTO_OP

The security function SF_CRYPTO_OP provides to users the cryptographic support to perform encryption/decryptiona and signature/verification.

The cryptographic algorithm supported are:

- DES – ECB in (encryption/decryption)
- DES – CBC in (encryption/decryption)
- Triple DES – ECB in (encryption/decryption) with 16, 24 bytes of key
- Triple DES – CBC in (encryption/decryption) with 16, 24 bytes of key
- AES – ECB in (encryption/decryption) with 128, 192, 256 bytes of key
- AES – CBC in (encryption/decryption) with 128, 192, 256 bytes of key
- RSA - both with key in standard and CRT mode. Key length supported 512, 768, 1024 bits.

The security function provides also means for the signature calculation and verification and for the message digest.

The padding algorithms are based on the standards ISO 9797 and PKCS#1. The security function SF_CRYPTO_OP uses the random number generator in order to pad input data according the standard PKCS#1 v1.5. The random number generator is based on the hardware and is not part of the TOE.

In order to preserve secret values all the buffer used in the calculation are cleared after the execution of a crypto operation.

The security function rely on the operating system dedicated software and hence on crypto library in order to perform the elementary cryptographic operations like simple DES, simple AES and modular exponentiation for RSA.

This function has no SOF claims.

7.1.4 SECURITY FUNCTION SF_FIREWALL

Enforce the security model of Java Card and manage the inter application resource sharing in a secure a controlled way. Enforce integrity check on the object headers and fileds.

Its main responsibilities are:

- to check inter-context access;
- to manage JCRE Entry Point Object and Global Array so to control the applets access to system resources;
- to control Shareable access so to permit inter applet resources sharing in a secure and aware way;

- to check temporary references storing; Temporary JCRE Entry Point Object references cannot be stored in any static fields.
- to handle currently active context.

This function has no SOF claims since it does not include any permutational or probabilistic mechanism.

7.1.5 SECURITY FUNCTION SF_OBJ_MNGT

SF_OBJ_MNGT main responsibility are:

- object creation with enforcement of consistent values for the security attributes;
- transient object creation with enforcement of consistent values for the security attributes;
- initializes the allocated resources with the appropriate default values;
- object deletetion on demand;
- Clear the memory allocated to a transients at their de-allocation;
- check the integrity of objects (header and fields);
- throw exception if the resource allocation is not possible since unavailability of resource.

This function has no SOF claims since it does not include any permutational or probabilistic mechanism.

7.1.6 SECURITY FUNCTION SF_PIN

This security function is related to all the operation concerning to PIN objects.

In particular SF_PIN :

- provides means to perform PIN Verification;
- decrease automatically the try check counter of PINs in case of pin failure;
- provides the functionality to update PIN value and the try counter.

PIN verification procedure consists in the comparson of the PIN provided by the user application requesting the verification procedure with the PIN stored into a PIN object. Pin verification is performed by means of a secure algorithm so to assure that the comparison between the secret value and the presented PIN takes always the same time independently from the values involved in the comparison. This prevents attacks based on timing analysis. This security function guarantee the unobservability of the operation performed.

The try check counter is managed so to avoid missing in the update if failure occurs during PIN verification.

The security function assures the integrity of the stored PINs.

One permutational mechanisms has been found in this SF: the PIN verification procedure;

This function has no SOF claims since the protection of the integrity of PIN objects and the non-observabilty of the verification procedure are the security functions of SF_PIN and not the PIN verification itself.

7.1.7 SECURITY FUNCTION SF_POST

This Security Function is in charcge to perform the power on self test.

The following test are performed:

- the integrity of the card lifecycle information;
- EEPROM writing operation failure
- DES test;
- AES test;
- RSA test;

This security function relies on the SF_CRYPTO_OP for the cryptographic calculation.

This function has no SOF claims since it does not include any permutational or probabilistic mechanism.

7.1.8 SECURITY FUNCTION SF_RMI

SF_RMI controls all the operations concerning the Java Card Remote Method Invocation.

It provides the mechanisms for a client application running on the CAD platform to invoke a method on a remote object on the card.

SF_RMI grants access to Remote Objects, according to the lifetime of their corresponding remote references, and controls the information flow concerning remote object references.

Moreover, it allocates array parameters of remote method invocations as global arrays on the card and restrintcs the storage of references to those arrays.

This function has no SOF claims since it does not include any permutational or probabilistic mechanism.

7.1.9 SECURITY FUNCTION SF_TRANSACTION

SF_TRANSACTION controls all the operations concerning "persistent memory" modification in order to guarantee the coherence of the sensible data if a failure occurs during the update.

This SF has two main responsibilities:

- Grant the atomic updates of class fields, instance fields and array elements;
- Give to the TOE the support for Java Card transactional mechanism;

It relies on dedicated buffers in EEPROM.

All the EEPROM data stored in EEPROM updated are either completely update or its values is completely maintained as at the beginning of the transaction. This guarantee the coherence of the sensible data stored in EEPROM if a failure occurs during the update.

This function has no SOF claims since it does not include any permutational or probabilistic mechanism.

7.2 ASSURANCE MEASURES

Table 40 reports the link between the assurance measures, the assuance requirements and the documentation containing the evidences that assurance measures fulfill the assurance requirements.

Assurance Measure	Assurance Requiement	Documents Satisfing the Assurance Requirements
AM_ACM	ACM_AUT.1	STRSME650-B - ACM_AUT.1, ACM_CAP.4, ACM_SCP.2 Mokard Safe 2.2 Software Configuration Management
	ACM_CAP.4	
	ACM_SCP.2	
AM_ADO	ADO_DEL.2	STRSME652-B - Mokard Safe 2.2: Delivery and Operation ADO_DEL.2;
	ADO_IGS.1	STRSME658-B - Mokard Safe 2.2: Initialization Geneation and Start up ADO_IGS.1;
AM_ADV	ADV_FSP.2	STRSME610-B - Mokard Safe 2.2: The Functional Specifications ADV_FSP.2;
	ADV_HLD.2	STRSME611-B - Mokard Safe 2.2: The High Level Design ADV_HLD.2;
	ADV_IMP.2	STRSME612-B - Mokard Safe 2.2: Implementation Representation ADV_IMP.2;
	ADV_LLD.1	STRSME613-B - Mokard Safe 2.2: The Low Level Design ADV_LLD.1;
	ADV_RCR.1	Included into: STRSME610-B - Mokard Safe 2.2: The Functional Specifications ADV_FSP.2; STRSME611-B - Mokard Safe 2.2: The High Level Design ADV_HLD.2; STRSME612-B - Mokard Safe 2.2: Implementation Representation ADV_IMP.2; STRSME613-B - Mokard Safe 2.2: The Low Level Design ADV_LLD.1;
	ADV_SPM.1	STRSME632-B - Mokard Safe 2.2: The security Policy Model ADV_SPM.1;
AM_AGD	AGD_ADM.1	STRSME630-B - Mokard Safe 2.2: The User and Administrator Guidance AGD_USR.1 – AGD_ADM.1;
	AGD_USR.1	STRSME630-B - Mokard Safe 2.2: The User and Administrator Guidance AGD_USR.1 – AGD_ADM.1;
AM_ALC	ALC_DVS.1	STRSME633-B – The Development Environment Security ALC_DVS.1;

	ALC_LCD.1	STRSME659-B "Mokard Safe2.2 ALC_LCD - TOE Life Cycle"
	ALC_TAT.1	STRSME653-B - Mokard Safe 2.2 Tool and Techniques ALC_TAT.1;
AM_ATE	ATE_COV.2	STRSME657-B - Mokard Safe2.2's TOE - Software Test Plan
	ATE_DPT.1	
	ATE_FUN.1	
	ATE_IND.2	
AM_AVA	AVA_MSU.2	STRSME656-B - Mokard Safe 2.2: Evaluation of Misuse AVA_MSU.2;
	AVA_SOF.1	STRSME655-B - Mokard Safe 2.2: Strenght of Function Analysis AVA_SOF.1;
	AVA_VLA.3	STRSME654-B - Mokard Safe 2.2: The Vulnerability Analysis AVA_VLA.3;

8 TABLE 40 – ASSURANCE MEASURES EVIDENCESPP CLAIMS

8.1 PP REFERENCE

This Security Target is compliant with the Protection Profile "Java Card System Protection Profile Collection" version 1.0b [JCSPPC] in the configuration Java Card Standard 2.2 registered at the French Certification Body DCSSI under the number PP/0305.

8.2 PP TAILORING

No tailoring has been performed to the ST.

8.3 PP ADDITION

The following additions have been performed to the Protection Profile PP/0305 in this Security Target:

- ***FMT_SMF.1.1/JCRE*** , ***FMT_SMF.1.1/ADEL*** , ***FMT_SMF.1.1/JCRMI*** , ***FMT_SMF.1.1/CM*** for the TOE;
- ***FMT_SMR.1.1/CMGR***, ***FMT_SMF.1.1/BCV*** for the IT Environment.

9 Rationale

This chapter presents the evidence to claims that th Security Target is a complete and cohesive set of requirements and that a conformant TOE would provide an effective set of IT security countermeasures within the security environment.

9.1 SECURITY OBJECTIVES RATIONALE

As this Security Target is conformant with the PP/0305 [JCSPPC] , its rationales are also applicable to the Security Target. Moreover, the Security Target does not add any element, such as assumption, threats or organizational security policies, in the TOE environment. Therfore , the security objective rationale provided in PP/0305 [JCSPPC] is sufficient.

9.1.1 Threats Related to Security Objectives

All the security objectives fixed for the TOE and its environment contribute to counter some threat on the assets. In order to provide evidence that all threats are actually prevented by some combination of security objectives, the presentation is oriented by the threats.

T.PHYSICAL — Covered by *OE.*SCP.*IC*. Physical protections rely on the underlying platform and are therefore an environmental issue.

CONFIDENTIALITY & INTEGRITY

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

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

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

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

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

Concerning the confidentiality and integrity of application sensitive data, as applets may need to share some data or communicate with the CAD, cryptographic functions are required to actually protect the exchanged information (*O.CIPHER*). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the applets to use them. Keys and PIN's are particular cases of an application's sensitive data[9] that ask for appropriate management (*O.KEY-MNGT*, *O.PIN-MNGT*, *O.TRANSACTION*). If the PIN class of the Java Card API is used, the objective (*O.FIREWALL*) is also concerned.

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

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

IDENTITY USURPATION

T.SID.1

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

In this configuration, usurpation of identity resulting from a malicious installation of an applet on the card is covered by the objective *O.INSTALL*.

The installation parameters of an applet (like its name) are loaded into a global array that is also shared by all the applications. The disclosure of those parameters (which could be used to impersonate the applet) is countered by the objective (*O.SHRD_VAR_CONFID*) and (*O.SHRD_VAR_INTEG*).

T.SID.2

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

The objective *O.INSTALL* contributes to counter this threat for what relates to the critical phase of applet installation (because the installer may have special rights).

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

UNAUTHORIZED EXECUTIONS

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

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

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

An application cannot download its own native code on the card, see the objective *OE.APPLET*, which also contributes to enforce the objective countering this threat (*O.NATIVE*).

DENIAL OF SERVICE

T.RESOURCES An attacker prevents correct operation of the Java Card System through consumption of some resources of the card. This is directly countered by objectives on resource-management (*O.RESOURCES*) for runtime purposes and good working order (*O.OPERATE*) in a general manner.

In this configuration, consumption of resources during installation and other card management operations are covered, in case of failure, by *O.INSTALL*.

Note that, for what relates to CPU usage, the Java Card platform is single–threaded and it is possible for an ill–formed application (either native or not) to monopolize the CPU. However, a smart card can be physically interrupted (card removal or hardware reset) and most CADs implement a timeout policy that prevent them from being blocked should a card fails to answer. That point is out of scope of this PP, though.

MODIFICATIONS OF THE SET OF APPLICATIONS

T.INSTALL The attacker fraudulently **installs** an `applet` on the card post issuance. This threat is covered by the *O.INSTALL* and *O.LOAD* security objectives.

INTEGRITY AND INSTALLATION

T.INTEG-APPLI-CODE.2 The attacker modifies (part of) its own or another application code when an application `package` is transmitted to the card for installation. In this configuration the integrity of a package's code is covered by the objective *O.LOAD*.

T.INTEG-APPLI-DATA.2 The attacker modifies (part of) the initialization data contained in an application `package` when the package is transmitted to the card for installation. In this configuration the integrity of a package's code is covered by the objective *O.LOAD*.

UNAUTHORIZED EXECUTIONS

T.EXE-CODE-REMOTE The *O.REMOTE* security objective contributes to prevent the invocation of a method that is not supposed to be accessible from outside the card.

CARD MANAGEMENT

T.DELETION This threat is covered by the *O.DELETION* security objective.

OBJECT DELETION

T.OBJ-DELETION This threat is covered by the *O.OBJ-DELETION* security objective.

The objective *OE.CARD-MANAGEMENT* supports *OE.VERIFICATION* and contributes to cover all the threats on confidentiality and integrity of code and data, the *T.*INSTALL threat, the *T.DELETION* threat and the **T.INTEG-APPLI-CODE.2** and **T.INTEG-APPLI-DATA.2** threats. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.*SID*.1*.

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

	O.INSTALL	*O.LOAD*	*OE.VERIFICATION*	*OE.CARD-MANAGEMENT*	*OE.APPLET*	*O.SHRD_VAR_INTEG*	*O.SHRD_VAR_CONFID*	*O.FIREWALL*	*O.NATIVE*	*O.OPERATE*	*O.ALARM*	*O.REALLOCATION*	*O.RESOURCES*	*O.SID*	*OE.SCP.IC*	*OE.SCP.RECOVERY*	*OE.SCP.SUPPORT*	*O.CIPHER*	*O.KEY-MNGT*	*O.PIN-MNGT*	*O.TRANSACTION*	*O.DELETION*	*O.REMOTE*	*O.OBJ-DELETION*
T.PHYSICAL															X									
*T.*CONFID-JCS-CODE *T.*INTEG-APPLI-CODE *T.*INTEG-JCS-CODE			X	X																				
*T.*CONFID-JCS-DATA *T.*INTEG-JCS-DATA			X	X				X		X	X			X		X	X							
*T.*CONFID-APPLI-DATA			X	X			X	X		X	X	X		X		X	X	X	X	X	X			
*T.*INTEG-APPLI-DATA			X	X		X		X		X	X	X		X		X	X	X	X	X	X			
*T.*SID.*1*	X			X		X	X	X						X										
*T.*SID.*2*	X							X		X				X		X	X							
T.EXE-CODE.1			X					X																
T.EXE-CODE.2			X																					
*T.*NATIVE			X		X				X															
*T.*RESOURCES	X									X			X			X	X							
*T.*INSTALL	X	X		X																				
T.INTEG-APPLI-CODE.2		X		X																				
T.INTEG-APPLI-DATA.2		X		X																				
T.DELETION				X																		X		
T.EXE-CODE-REMOTE																							X	
T.OBJ-DELETION																								X

Table 41: Threats rationale

9.1.2 Assumptions Related to Security Objectives

This section relates the security objectives to be achieved by this configuration to the assumptions made on the TOE and its environment.

In this configuration all the security objectives directly or indirectly depend on the behavior of the native code embedded on the card. This trusted native code is not subject to change during the lifetime of the card. The objective *OE.NATIVE* ensures that the environmental assumption *A.NATIVE* is upheld. The objective *OE.APPLET* covers the assumption *A.APPLET*, and contributes to the enforcement of the objective *O.NATIVE* in the presence of post-issuance downloaded applications. The objective *OE.VERIFICATION* upholds the assumption *A.VERIFICATION*.

Table 42 provides a mapping of security objectives to the assumptions made on the environment of the TOE.

	OE.NATIVE	OE.APPLET	OE.VERIFICATION
A.NATIVE	X		
A.APPLET		X	
A.VERIFICATION			X

Table 42: Assumptions rationale

9.1.3 Organizational Policies Related to Security Objectives

Only one organizational security policy, *OSP.VERIFICATION*, has been defined for this configuration. This policy is covered by the security objective of the environment *OE.VERIFICATION*.

9.2 SECURITY REQUIREMENTS RATIONALE

As this Security Target is conformant with the PP/0305 [JCSPPC] , its rationales are also applicable to the Security Target. Moreover, the Security Target does not add any element, such as assumption, threats or organizational security policies, in the TOE environment. Therfore , the security the security functional requirements rationale provided in PP/0305 [JCSPPC] is sufficient.

This section is devoted to demonstrate that the set of security requirements (both on the TOE and on the environment) is suitable to meet security objectives. The presentation follows the same structure as §5.1, listing the requirements that are related to each objective of each configuration.

The following conventions shall are used throughout this section:

- In the text of the rationales there shall be explicit references to (access and information flow) control policies, as contributing to meet certain security objectives. These references shall be associated to the principal security components by means of which those policies are defined, FDP_ACC and FDP_ACF in the case of control policies; FDP_IFC and FDP_IFF in the case of information flow ones, as well as to all the SFRs on which the afore mentioned components depend. The rationale tables, on the contrary, shall make it explicit which security objectives the components involved in those policies contribute to meet.

- The name of a SFR class component shall be used to make reference to (all) the iterations of that component which are present in a configuration. By present in a configuration it must be understood as belonging to one of the groups included in that configuration.

- A reference to a particular iteration of a SFR component shall be denoted as Component_Name/Label, where Label shall be the name of the TOE component.

9.2.1 TOE Security Requirements Rationale

In the context of this rationale the *FIREWALL access control policy* is the one specified in the group LCG. The references to the components *FDP_ACC.2/FIREWALL*, *FDP_ACF.1/FIREWALL* and *FMT_MSA.1/JCRE* must be understood as denoting the definitions of those components as provided in the group LCG.

IDENTIFICATION	
O.SID	Subjects' identity is AID-based (applets, packages), and is met by ***FDP_ITC.2***, ***FMT_MSA.1***, FMT_SMF.1, ***FMT_MSA.3***, ***FMT_MTD.1*** and ***FMT_MTD.3***. Additional support includes ***FPT_RVM.1*** and ***FPT_SEP.1***. Lastly, installation procedures ensure protection against forgery (the AID of an applet is under the control of the TSFs) or re-use of identities (***FIA_UID.2***, ***FIA_USB.1***).

APPLET MANAGEMENT	
O.INSTALL	This objective specifies that installation of applets must be secure. Security attributes of installed data are under the control of the *FIREWALL* access control *policy* (***FDP_ITC.2***) and the TSFs are protected against possible failures of the installer (*FPT_FLS.1/Installer*, *FPT_RCV.3*).
O.LOAD	This objective specifies that the loading of a package into the card must be secure. Evidence of the origin of the package is enforced (*FCO_NRO.2*) and the integrity of the corresponding data is under the control of the ***PACKAGE LOADING information flow*** policy (*FDP_IFC.2/CM*, *FDP_IFF.1/CM)* and *FDP_UIT.1*. Appropriate identification (*FIA_UID.1/CM*) and transmission mechanisms are also enforced *(FTP_ITC.1)*.
O.DELETION	This objective specifies that applet and package deletion must be secure. The non-introduction of security holes is ensured by the *ADEL access control* ***policy*** (*FDP_ACC.2/ADEL*, *FDP_ACF.1/ADEL)*. The integrity and confidentiality of data that does not belong to the deleted applet or package is a by-product of this policy as well. Non-accessibility of deleted data is met by *FDP_RIP.1/ADEL* and the TSFs are protected against possible failures of the deletion procedures (*FPT_FLS.1/ADEL*, *FPT_RCV.3* (see application note)). The functional requirements of the class ***FMT*** included in the group *ADELG* also contribute to meet this objective.

EXECUTION

O.OPERATE

The TOE is protected in various ways against applets' actions (***FPT_RVM.1***, ***FPT_SEP.1***, ***FPT_TDC.1*** the ***FIREWALL*** *access control policy* (*FDP_ACC.2*, *FDP_ACF.1*), and is able to detect and block various failures or security violations during usual working (***FPT_FLS.1***, ***FAU_ARP.1***). Startup of the TOE is covered by ***FPT_TST.1***, and indirectly by ***FPT_AMT.1*** (this latter defined in group *SCPG* .

Its security-critical parts and procedures are also protected: safe recovery from failure is ensured (*FPT_RCV.3*), applets' installation may be cleanly aborted (**FDP_ROL.1**), communication with external users and their internal subjects is well-controlled (***FDP_ITC.2***, ***FIA_ATD.1***, ***FIA_USB.1***) to prevent alteration of TSF data (also protected by components of the FPT class).

Almost every objective and/or functional requirement indirectly contributes to this one too.

O.RESOURCES

The TSFs detects stack/memory overflows during execution of applications (***FAU_ARP.1***, ***FRU_RSA.1***, **FPT_FLS.1**). Failed installations are not to create memory leaks (**FDP_ROL.1**, *FPT_RCV.3*) as well. Memory management is controlled by the TSF (**FMT_MTD.1**, ***FMT_MTD.3***, *FMT_SMR.1*) and is only accessible to user-applications through the API (***FPT_RVM.1***).

O.FIREWALL

This objective is met by the ***FIREWALL*** *access control policy* (*FDP_ACC.2*, *FDP_ACF.1*), the *JCVM information flow control policy* (***FDP_IFF.1***, ***FDP_IFC.1***), the *JCRMI access control policy* (*FDP_ACC.2/JCRMI*, *FDP_ACF.1/JCRMI*) and the functional requirements ***FPT_RVM.1***, ***FPT_SEP.1*** and **FDP_ITC.2**. The functional requirements of the class ***FMT*** also indirectly contribute to meet this objective.

O.NATIVE

The JCVM is the machine running the bytecode of the applets (***FPT_RVM.1***). These can only be linked with API methods or other packages already on the card. This objective mainly relies on the environmental objectives ***OE.NATIVE*** and *OE.APPLET*, which uphold the assumptions *A.NATIVE* and *A.APPLET* respectively.

O.REALLOCATION

The security objective is satisfied by ***FDP_RIP.1***, which imposes that the contents of the re-allocated block shall always be cleared before delivering the block.

O.SHRD_VAR_CONFID

Only arrays can be designated as global, and the only global arrays required in the Java Card API are the APDU buffer and the byte array input parameter (`bArray`) to an applet's install method. The clearing requirement of those arrays is met by *FDP_RIP.1* (***FDP_RIP.1.1/APDU*** and ***FDP_RIP.1.1/bArray*** respectively). The *JCVM information flow control policy* (***FDP_IFF.1***, ***FDP_IFC.1***) prevents an application from keeping a pointer to

a shared buffer, which could be used to read its contents when the buffer is being used by another application.

Protection of the array parameters of remotely invoked methods, which are global as well, is covered by the general initialization of method parameters (***FDP_RIP.1***).

O.SHRD_VAR_INTEG This objective is met by the *JCVM information flow control policy* (***FDP_IFF.1***, ***FDP_IFC.1***), which prevents an application from keeping a pointer to the input/output buffer of the card, or any other global array that is shared by all the applications. Such a pointer could be used to access and modify it when the buffer is being used by another application.

SERVICES

O.ALARM This objective is met by ***FPT_FLS.1*** and ***FAU_ARP.1*** (see application notes).

O.TRANSACTION Directly met by **FDP_ROL.1** and *FDP_RIP.1* (more precisely, by the element ***FDP_RIP.1.1/ABORT***).

Transactions are provided to applets as Java Card class libraries.

O.CIPHER This objective is directly related to ***FCS_CKM.1***, ***FCS_CKM.2***, ***FCS_CKM.4***, ***FCS_COP.1***, Another important SFR is ***FPR_UNO.1***, the observation of the cryptographic operations may be used to disclose the keys.

The associated security functions are not described herein. They are provided to applets as Java Card class libraries (see the class **`javacardx.crypto.Cipher`** and the package **`javacard.security`**).

O.PIN-MNGT This objective is ensured by ***FDP_RIP.1***, ***FPR_UNO.1***, ***FDP_ROL.1*** and ***FDP_SDI.2*** functional requirements. The security functions behind these are implemented by API classes. The firewall security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the objects.

O.KEY-MNGT This relies on the same functional requirements as *O.CIPHER*, plus ***FDP_RIP.1*** and ***FDP_SDI.2*** as well.

O.REMOTE The access to the TOE's internal data and the flow of information from the card to the CAD required by the JCRMI service is under control of the *JCRMI access control policy* (*FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI*) and the ***JCRMI*** *information flow* control policy (*FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI*). The functional requirements of the class ***FMT*** included in the group *RMIG* also contribute to meet this objective.

OBJECT DELETION

O.OBJ-DELETION This objective specifies that deletion of objects is secure. The objective is met by the functional requirements *FDP_RIP.1/ODEL* and *FPT_FLS.1/ODEL*.

	FAU_ARP.1	FCS_CKM.1	FCS_CKM.2	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FDP_ROL.1	FDP_SDI.2	FIA_ATD.1	FIA_UID.2	FIA_USB.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_MTD.3	FMT_SMF.1	FMT_SMR.1	FPR_UNO.1	FPT_FLS.1	FPT_RVM.1	FPT_SEP.1	FPT_TDC.1	FPT_TST.1
O.ALARM	X																								X				
O.CIPHER		X	X	X	X	X																		X					
O.FIREWALL							X	X	X	X							X	X	X	X	X	X	X			X	X		
O.KEY-MNGT		X	X	X	X	X					X		X											X					
O.NATIVE																										X			
O.OPERATE	X						X	X				X		X		X									X	X	X	X	X
O.PIN-MNGT							X	X			X	X	X											X					
O.RESOURCES	X											X								X	X		X		X	X			
O.SID														X	X	X	X		X	X	X	X				X	X		
O.TRANSACTION											X	X																	
O.SHRD_VAR_CONFID									X	X	X																		
O.SHRD_VAR_INTEG									X	X																			
O.REALLOCATION											X																		

	FCO_NRO.2	FDP_IFC.2	FDP_IFF.1	FDP_ITC.2	FDP_UIT.1	FIA_UID.1	FPT_FLS.1	FPT_RCV.3	FRU_RSA.1	FTP_ITC.1
O.INSTALL				X			X	X		
O.LOAD	X	X	X		X	X				X
O.SID				X						
O.OPERATE				X				X		
O.RESOURCES								X	X	
O.FIREWALL				X						

	FDP_ACC.2	*FDP_ACF.1*	*FDP_IFC.1*	*FDP_IFF.1*	*FDP_RIP.1*	*FMT_MSA.1*	*FMT_MSA.3*	*FMT_SMF.1*	*FMT_REV.1*	*FMT_SMR.1*	*FPT_FLS.1*	*FPT_RCV.3*
O.DELETION	X	X			X	X	X	X		X	X	X
O.OBJ-DELETION					X						X	
O.REMOTE	X	X	X	X		X	X	X	X	X		
O.FIREWALL									X			

Table 43: Security requirements rationale

9.2.1.1 IT Environment Security Requirements Rationale

The environmental objective *OE.VERIFICATION*, which is satisfied by IT procedural means, is met by the SFRs of the group *BCVG* (§6.3.3).

The environmental objective *OE.APPLET* might be also satisfied by IT procedural means. The IT verification that a post-issuance loaded applet contains no native code could be carried out as a part of the verification of how well the CAP file is formed. This verification has been associated in the group *BCVG* (§6.3.3) to the requirement of secure security attributes, expressed by the component ***FMT_MSA.2*** (see application note at pp. 99).

The environmental objective *OE.CARD-MANAGEMENT*, which is satisfied by IT procedural means, is met by the SFRs of the group *CMGRG* (§6.3.2).

All the security functional requirements to which this section makes reference from now on are those specified in the group SCPG.

The components *FPT_RCV.3* and *FPT_RCV.4* are used to support the objective *OE.*SCP.*SUPPORT* and *OE.*SCP.*RECOVERY* to assist the TOE to recover in the event of a power failure. If the power fails or the card is withdrawn prematurely from the CAD the operation of the TOE may be interrupted leaving the TOE in an inconsistent state.

OE.SCP.RECOVERY This objective is met by the components *FPT_FLS.1*, *FPT_RCV.3* and *FRU_FLT.1*.

OE.SCP.SUPPORT This objective is met by the components *FPT_SEP.1* (no bypassing TSF), *FPT_AMT.1*, *FPT_RCV.3*, *FPT_RCV.4* and *FPT_RVM.1*.

OE.SCP.IC This objective is met by the component *FPT_PHP.3*.

	FPT_AMT.1	*FPT_FLS.1*	*FPT_PHP.3*	*FPT_RCV.3*	*FPT_RCV.4*	*FPT_RVM.1*	*FPT_SEP.1*	*FRU_FLT.1*
OE.SCP.RECOVERY		X		X				X
OE.SCP.SUPPORT	X			X	X	X	X	
OE.SCP.IC			X					

Table 44: Security requirements rationale

9.2.1.2 Security Functional Requirements Dependencies

The TOE assurance requirements dependencies for level EAL4 are completely fulfilled.

The functional requirements dependencies for the TOE are not completely fulfilled. The KOs in the following table corresponds to unsatisfied dependencies that are explained and justified in the rationale that appears below the table.

SFR	Dependency	Status
FAU_ARP.1/JCS	(FAU_SAA.1)	KO : FAU_SAA.1 is not satisfied
FCO_NRO.2/CM	(FIA_UID.1)	OK: FIA_UID.1/CM
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: ***FCS_CKM.2***, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.2	(FDP_ITC.1 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: ***FCS_CKM.1***, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.3	(FDP_ITC.1 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: ***FCS_CKM.1***, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.4	(FDP_ITC.1 or FCS_CKM.1) and (FMT_MSA.2)	OK: ***FCS_CKM.1***, FMT_MSA.2/JCRE
FCS_COP.1	(FDP_ITC.1 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: ***FCS_CKM.1***, FCS_CKM.4, FMT_MSA.2/JCRE
FDP_ACC.1/CMGR	(FDP_ACF.1)	OK: FDP_ACF.1/CMGR
FDP_ACC.1/ADEL	(FDP_ACF.1)	OK:FDP_ACF.1/ADEL
FDP_ACC.1/FIREWALL	(FDP_ACF.1)	OK: FDP_ACF.1/FIREWALL
FDP_ACC.1/JCRMI	(FDP_ACF.1)	OK: FDP_ACF.1/JCRMI

SFR	Dependency	Status
FDP_ACF.1/ADEL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/ADEL, FMT_MSA.3/ADEL
FDP_ACF.1/CMGR	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/CMGR, FMT_MSA.3/CMGR
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/FIREWALL, FMT_MSA.3/FIREWALL
FDP_ACF.1/JCRMI	(FDP_ACC.1) and (FMT_MSA.3)	OK FDP_ACC.1/JCRMI, FMT_MSA.3/JCRMI
FDP_IFC.1/JCRMI	(FDP_IFF.1)	OK: FDP_IFF.1/JCRMI
FDP_IFC.1/JCVM	(FDP_IFF.1)	OK: FDP_IFF.1/JCVM
FDP_IFC.1/BCV	(FDP_IFF.1)	OK: FDP_IFF.2/BCV
FDP_IFC.1/CM	(FDP_IFF.1)	OK: FDP_IFF.1/CM
FDP_IFF.1/CM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/CM, FMT_MSA.3/CM
FDP_IFF.1/JCRMI	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCRMI, FMT_MSA.3/JCRMI
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCVM, FMT_MSA.3/FIREWALL
FDP_IFF.2/BCV	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/BCV, FMT_MSA.3/BCV
FDP_ITC.2	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1) and (FPT_TDC.1)	OK: **FPT_TDC.1**, FDP_IFC.1/CM, FTP_ITC.1/CM
FDP_ROL.1	None	OK
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	OK: FDP_ACC.1/FIREWALL, FDP_IFC.1/JCVM
FDP_SDI.2	None	OK
FDP_UIT.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1)	OK: FDP_IFC.1/CM, FTP_ITC.1/CM
FIA_ATD.1/AID	None	OK
FIA_UID.1/CM	None	OK
FIA_UID.1/CMGR	None	OK
FIA_UID.1/AID	None	OK
FIA_USB.1	(FIA_ATD.1)	OK: FIA_ATD.1/AID
FMT_MSA.1/ADEL	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_ACC.1/ADEL, FMT_SMR.1/ADEL, FMT_SMF.1/ADEL
FMT_MSA.1/BCV	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_IFC.1/BCV, FMT_SMR.1/BCV, FMT_SMF.1/BCV
FMT_MSA.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_IFC.1/CM, FMT_SMR.1/CM, FMT_SMF.1/CM

SFR	Dependency	Status
FMT_MSA.1/CMGR	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_ACC.1/CMGR, FMT_SMR.1/CMGR, FMT_SMF.1/CMGR
FMT_MSA.1/EXPORT FMT_MSA.1/JCRMI FMT_MSA.1/REM-REFS	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_IFC.1/JCRMI, FMT_SMR.1/JCRMI, FMT_SMF.1/JCRMI
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and FMT_SMF.1	OK: FDP_ACC.1/FIREWALL, FDP_IFC.1/JCVM, FMT_SMR.1/JCRE, FMT_SMF.1/JCRE
FMT_MSA.2/JCRE	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_ACC.1/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.2/BCV	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_IFC.1/BCV, FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/ADEL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/ADEL, FMT_SMR.1/ADEL
FMT_MSA.3/BCV	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/CM	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CM, FMT_SMR.1/CM
FMT_MSA.3/CMGR	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CMGR, FMT_SMR.1/CMGR
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.3/JCRMI	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MTD.1/JCRE	(FMT_SMR.1) and FMT_SMF.1	OK: FMT_SMR.1/JCRE, FMT_SMF.1/JCRE
FMT_MTD.3	(ADV_SPM.1) and (FMT_MTD.1)	OK: FMT_MTD.1/JCRE
FMT_REV.1/JCRMI	(FMT_SMR.1)	OK: FMT_SMR.1/JCRMI
FMT_SMF.1.1/ADEL	None	OK
FMT_SMF.1.1/BCV	None	OK
FMT_SMF.1.1/CM	None	OK
FMT_SMF.1.1/CMGR	None	OK
FMT_SMF.1.1/JCRE	None	OK
FMT_SMF.1.1/JCRMI	None	OK
FMT_SMR.1/ADEL	(FIA_UID.1)	KO: FIA_UID.1
FMT_SMR.1/BCV	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/CM	(FIA_UID.1)	OK: FIA_UID.1/CM
FMT_SMR.1/CMGR	(FIA_UID.1)	OK: FIA_UID.1/CMGR
FMT_SMR.1/JCRE	(FIA_UID.1)	OK: FIA_UID.1/AID
FMT_SMR.1/Installer	(FIA_UID.1)	KO: FIA_UID.1

SFR	Dependency	Status
FMT_SMR.1/JCRMI	(FIA_UID.1)	OK: FIA_UID.1/AID
FPR_UNO.1	None	OK
FPT.PHP.3/SCP	None	OK
FPT_AMT.1/SCP	None	OK
FPT_FLS.1/ADEL	(ADV_SPM.1)	OK
FPT_FLS.1/Installer	(ADV_SPM.1)	OK
FPT_FLS.1/JCS	(ADV_SPM.1)	OK
FPT_FLS.1/ODEL	(ADV_SPM.1)	OK
FPT_FLS.1/SCP	(ADV_SPM.1)	OK
FPT_RCV.3/Installer	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK: **FPT_TST.1**
FPT_RCV.3/SCP	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK:**FPT_TST.1**
FPT_RCV.4/SCP	(ADV_SPM.1)	OK
FPT_RVM.1	None	OK
FPT_RVM.1/SCP	None	OK
FPT_SEP.1	None	OK
FPT_SEP.1/SCP	None	OK
FPT_TDC.1	None	OK
FPT_TST.1	(FPT_AMT.1)	OK: FPT_AMT.1/SCP
FRU_FLT.1/SCP	(FPT_FLS.1)	OK: FPT_FLS.1/SCP
FRU_RSA.1/Installer	None	OK
FRU_RSA.1/BCV	None	OK
FTP_ITC.1/CM	None	OK

Table 45: Functional Requirement Dependencies

FAU_SAA.1

Potential violation analysis is used to specify the set of auditable events whose occurrence or accumulated occurrence held to indicate a potential violation of the TSP, and any rules to be used to perform the violation analysis. The dependency of FAU_ARP.1/JCS on this functional requirement assumes that a "potential security violation" is an audit event indicated by the FAU_SAA.1 component. The events listed in FAU_ARP.1/JCS are, on the contrary, merely self-contained ones (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The JCVM or other components of the TOE detect these events during their usual working order. Thus, in principle there would be no applicable audit recording in this framework. Moreover, no specification of one such recording is provided elsewhere. Therefore no set of auditable events could possibly be defined.

FIA_UID.1

This is required by the component ***FMT_SMR.1*** in group *InstG*. However, the role installer defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The installer does not "identify" itself with respect to the TOE, but is a part of it. Thus, here it

is claimed that this dependency can be left out. The reader may notice that the role is required because of the SFRs on management of TSF data and security attributes, essentially those of the firewall policy.

This is also required by the component ***FMT_SMR.1*** in group BCVG. See the explanation in the paragraph above (the role in this case is applet deletion manager).

This is also required by the component ***FMT_SMR.1*** in group BCVG. However, the role bytecode verifier defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The bytecode verifier does not "identify" itself with respect to the TOE, furthermore, it is part of the IT environment. Thus, here it is claimed that this dependency can be left out.

9.2.1.3 Rationale for Strength of Function High

The minimum strength of function level **SOF- must be high**.

The rationale of SOF-high is tha the TOE is intended to operate in open environments, where attackers can easily exploit vulnerabilities. According to the claimed intended usage of the TOE, hosting mCommerce, banking and signature applciations, it is very likely that it may represent a significant value and then constitute an attractive target for attacks. In some malicious usages of the TOE the statistical or probabilistic mechanisms in the TOE, for instance, may be subjected to analysis and attack in the normal course of operation.

PP/0305 set the minimum strength of function level to SOF-medium and leave to the card issuer the choice of high strength of function requirement on the base of the itended usage.

The strength of function level high is consistent with the vulnerability analysis level that has been specified (***AVA_VLA.3).***

9.2.1.4 Rationale for Assurance Level EAL4 augmented

The assurance level for this protection profile is EAL4 augmented. Augmentation results from the selection of the components ***AVA_VLA.3*** and ***ADV_IMP.2.***

9.2.1.4.1 Rationale for Assurance Level EAL4

EAL4 allows a developer to attain a reasonably high assurance level without the need for highly specialized processes and practices. It corresponds to a white box analysis and it can be considered as a reasonable level that can be applied to an existing product line without undue expense and complexity.

9.2.1.4.2 Rationale for Augmentation

The evaluation of the TOE may be performed, for instance, because the product hosts one or several sensitive applications, such as financial and health recording ones, which contain, represent, or provide access to valuable assets. In addition to that the TOE may not be directly under the control of trained and dedicated administrators.

AVA_VLA.3

As a result, it is imperative that the TOE vulnerabilities to be reviewed be drawn from a systematic search rather than strictly a manufacturer prepared identification list. Component ***AVA_VLA.3*** requires that such a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***.

AVA_VLA.3 has the following dependencies:

- ***ADV_FSP.1*** Informal functional specification
- ***ADV_HLD.2*** Security enforcing high-level design
- ***ADV_IMP.1*** Subset of the implementation of the TSF
- ***ADV_LLD.1*** Descriptive low-level design
- ***AGD_ADM.1*** Administrator guidance
- ***AGD_USR.1*** User guidance

All of these are met or exceeded in the EAL4 assurance package.

ADV_IMP.2

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement.

The assurance component ***ADV_IMP.2*** has been chosen because the evaluation of the TOE must ensure that its security functional requirements are completely and accurately addressed by the implementation representation of the TSF.

ADV_IMP.2 has the following dependencies:

- ***ADV_LLD.1*** Descriptive low-level design
- ***ADV_RCR.1*** Informal correspondence demonstration
- ***ALC_TAT.1*** Well-defined development tools

All of these are met or exceeded in the EAL4 assurance package.

9.2.1.5 Internal Consistency and Mutual Support

The purpose of this part of the Security Target rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given to this because:

- The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies are satisfied).

- The dependencies analysis for the functional requirements described in the section "Security Functional Requirements Dependencies" demonstrates mutual support and internal consistency between the functional requirements. That analysis also shows that the dependencies between functional and assurance requirements are also satisfied.

9.3 TOE SUMMARY SPECIFICATION RATIONALE

9.3.1 Security Functional Requirements Coverage Rationale

In this pharagraph a rationale between the security functional requirements and the security functions is reported. The aim of the rationale is to show that the combination of specified TOE IT security functions work together so as to satisfy the TOE security functional requirements.

Some security functional requirements are covered by all the TSFs. For these SFRs the rationale of coverage is provided once for all in the following sentences.

This paragraph is Incard property.

9.3.2 Security Assurance Requirements Coverage Rationale

In this pharagraph a rationale for the coverage of the security assurance requirements with the assurance measures is reported. The rationale give a justification that the assurance measures meet the TOE assurance requirements.

This paragraph is Incard property.

9.4 PP CLAIMS RATIONALE

The security objectives and the security requirements in this Security Target are the same as in the PP/0305. The Security Target does not add any element, such as assumption, threats or organizational security policies, in the TOE environment. There are no need of additional PP Claims rationale. This paragraph is not applicable at this Security Target

10APPENDIX: GLOSSARY

AID

Application identifier, an ISO-7816 data format used for unique identification of Java Card applications (and certain kinds of files in card file systems). The Java Card platform uses the AID data format to *identify* **applet**s *and* package**s**. AIDs are administered by the International Standards Organization (ISO), so they can be used as unique identifiers.

AIDs are also used in the security policies (see "Context" below): applets' AIDs are related to the selection mechanisms, packages' AIDs are used in the enforcement of the firewall. **Note:** although they serve different purposes, they share the same name space.

APDU

Application Protocol Data Unit, an ISO 7816-4 defined communication format between the card and the off-card applications. Cards receive requests for service from the CAD in the form of APDUs. These are encapsulated in Java Card System by the **`javacard.framework.APDU`** class ([JCAPI21]).

APDUs manage both the selection-cycle of the applets (through JCRE mediation) and the communication with the Currently selected applet.

APDU buffer

The APDU buffer is the buffer where the messages sent (received) by the card depart from (arrive to). The JCRE owns an APDU object (which is a JCRE Entry Point and an instance of the **`javacard.framework.APDU`** class) that encapsulates APDU messages in an internal byte array, called the APDU buffer. This object is made accessible to the Currently selected applet when needed, but any permanent access (out-of selection-scope) is strictly prohibited for security reasons.

applet

The name given to a Java Card technology-based user application. An applet is the basic piece of code that can be selected for execution from outside the card. Each applet on the card is uniquely identified by its AID.

applet deletion manager

The on-card component that embodies the mechanisms necessary to delete an applet or library and its associated data on smart cards using Java Card technology.

BCV

The bytecode verifier is the software component performing a static analysis of the code to be loaded on the card. It checks several kinds of properties, like the correct format of CAP files and the enforcement of the typing rules associated to bytecodes. If the component is placed outside the card, in a secure environment, then it is called an off-card verifier. If the component is part of the embedded software of the card it is called an on-card verifier.

CAD	Card Acceptance Device, or card reader. The device where the card is inserted, and which is used to communicate with the card.
CAP file	A file in the Converted applet format. A CAP file contains a binary representation of a package of classes that can be installed on a device and used to execute the package's classes on a Java Card virtual machine. A CAP file can contain a user library, or the code of one or more applets.
Class	In object-oriented programming languages, a class is a prototype for an object. A class may also be considered as a set of objects that share a common structure and behavior. Each class declares a collection of fields and methods associated to its instances. The contents of the fields determine the internal state of a class instance, and the methods the operations that can be applied to it. Classes are ordered within a class hierarchy. A class declared as a specialization (a subclass) of another class (its super class) inherits all the fields and methods of the latter. Java platform classes should not be confused with the classes of the functional requirements (FIA) defined in the CC.
Context	A context is an object-space partition associated to a package. Applets within the same Java technology-based package belong to the same context. The firewall is the boundary between contexts (see "Current context").
Current context	The JCRE keeps track of the current Java Card System context (also called "the active context"). When a virtual method is invoked on an object, and a context switch is required and permitted, the current context is changed to correspond to the context of the applet that owns the object. When that method returns, the previous context is restored. Invocations of static methods have no effect on the current context. The current context and sharing status of an object together determine if access to an object is permissible.
Currently selected applet	The applet has been selected for execution in the current session. The JCRE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command from the CAD with this applet's AID, the JCRE makes this applet the currently selected applet. The JCRE sends all APDU commands to the currently selected applet ([JCRE21] Glossary).
Default applet	The applet that is selected after a card reset ([JCRE21], §4.1).
Embedded Software	Pre-issuance loaded software.
Firewall	The mechanism in the Java Card technology for ensuring applet isolation and object sharing. The firewall prevents an applet in one context from unauthorized access to objects owned by the JCRE or by an applet in another context.

Installer

The installer is the on-card application responsible for the installation of applets on the card. It may perform (or delegate) mandatory security checks according to the card issuer policy (for bytecode-verification, for instance), loads and link packages (CAP file(s)) on the card to a suitable form for the JCVM to execute the code they contain. It is a subsystem of what is usually called "card manager"; as such, it can be seen as the portion of the card manager that belongs to the TOE.

The installer has an AID that uniquely identifies him, and may be implemented as a Java Card applet. However, it is granted specific privileges on an implementation-specific manner ([JCRE21], §10).

Interface

A special kind of Java programming language class, which declares methods, but provides no implementation for them. A class may be declared as being the implementation of an interface, and in this case must contain an implementation for each of the methods declared by the interface. (see also shareable interface).

JCRE

The Java Card runtime environment consists of the Java Card virtual machine, the Java Card API, and its associated native methods. This notion concerns all those dynamic features that are specific to the execution of a Java program in a smart card, like applet lifetime, applet isolation and object sharing, transient objects, the transaction mechanism, and so on.

JCRE Entry Point

An object owned by the JCRE context but accessible by any application. These methods are the gateways through which applets request privileged JCRE system services: the instance methods associated to those objects may be invoked from any context, and when that occurs, a context switch to the JCRE context is performed.

There are two categories of JCRE Entry Point Objects: Temporary ones and Permanent ones. As part of the firewall functionality, the JCRE detects and restricts attempts to store references to these objects.

JCRMI

Java Card Remote Method Invocation is the Java Card System, version 2.2, mechanism enabling a client application running on the CAD platform to invoke a method on a remote object on the card. Notice that in Java Card System, version 2.1.1, the only method that may be invoked from the CAD is the **process** method of the **applet** class.

Java Card System

The Java Card System: the JCRE (JCVM +API), the installer, and the on-card BCV (if the configuration includes one).

JCVM

The embedded interpreter of bytecodes. The JCVM is the component that enforces separation between applications (firewall) and enables secure data sharing.

logical channel	A logical link to an application on the card. A new feature of the Java Card System, version 2.2, that enables the opening of up to four simultaneous sessions with the card, one per logical channel. Commands issued to a specific logical channel are forwarded to the active applet on that logical channel.
Object deletion	The Java Card System, version 2.2, mechanism ensures that any unreferenced persistent (transient) object owned by the current context is deleted. The associated memory space is recovered for reuse prior to the next card reset.
Package	A package is a name space within the Java programming language that may contain classes and interfaces. A package defines either a user library, or one or more applet definitions. A package is divided in two sets of files: export files (which exclusively contain the public interface information for an entire package of classes, for external linking purposes; export files are not used directly in a Java Card virtual machine) and CAP files.
SCP	Smart Card Platform. It is comprised of the integrated circuit, the operating system and the dedicated software of the smart card.
Shareable interface	An interface declaring a collection of methods that an applet accepts to share with other applets. These interface methods can be invoked from an applet in a context different from the context of the object implementing the methods, thus "traversing" the firewall.
SIO	An object of a class implementing a shareable interface.
Subject	An active entity within the TOE that causes information to flow among objects or change the system's status. It usually acts on the behalf of a user. Objects can be active and thus are also *subjects* of the TOE.
Transient object	An object whose contents is not preserved across CAD sessions. The contents of these objects are cleared at the end of the current CAD session or when a card reset is performed. Writes to the fields of a transient object are not affected by transactions.
User	Any application interpretable by the JCRE. That also covers the packages. The associated subject(s), if applicable, is (are) an object(s) belonging to the **`javacard.framework.applet`** class.

End of Document