Java Card Protection Profile – Closed Configuration

December 2012
Version 3.0

Security Evaluations
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

This document has been prepared by:

Trusted Labs S.A.S
5, rue du Bailliage
78000 Versailles, France
http://www.trusted-labs.com

on behalf of Oracle Corporation.

For any correspondence on this document please contact the following organisations:

- **Oracle Corporation,**
  500 Oracle Parkway
  Redwood City,
  CA 94065 USA
  http://www.oracle.com
  seceval_us@oracle.com.

- **Secrétariat Général de la Défense Nationale**
  Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI)
  51, boulevard de Latour-Maubourg
  75700 Paris 07 SP, France
  http://www.ssi.gouv.fr/
  communication@ssi.gouv.fr

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 environment of a Java Card technology-enabled device 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 a Java Card technology-enabled system ("Java Card System"), compliant with Java Card platform specifications ("Java Card specifications"). These requirements should serve as a template for writing Common Criteria security targets of specific implementations of Java Card Systems. It therefore almost solely looks at the Java Card System from the security angle, a viewpoint that somewhat sets it apart from the usual functional documentation; that is, focused on what can happen rather than what should happen. It was written with critical real–life applications in mind. Accordingly, some aspects of the development and life–cycle of the applications are controlled, even though they are out of the scope of the software embedded on a Java Card platform.

In order to achieve a better understanding of the security issues of the Java Card System, this document provides a precise description of its background and possible environments, which is the first step to risk analysis. The division of duties and assignment of responsibilities among the several involved actors (both physical and IT components) leads to the definition of detailed security policies. Of course, there are cases where the choice is left to implementers; in all cases, risks and assets at stake are described to pave the way to security targets (ST).

One of the challenges of writing a Protection Profile for the Java Card technology is to address in a single description the wide range of choices offered (logical communication channels with the card, remote invocations of services, object deletion, among others), and the different security architectures that have been conceived so far (closed platforms, off-card verification of applications code, embedded verifiers, and so on).

The answer to this challenge is the definition of two main configurations corresponding to standard use-cases, the Closed Configuration and the Open Configuration. Each

configuration is addressed in a dedicated Protection Profile conformant to Common Criteria version 3.1.

The Closed and Open Configuration address versions 2.2.x and versions 3.0.x Classic Edition of Java Card Platform specifications. The Closed Configuration addresses Java Card Systems without post-issuance loading and installation of applets; the Open Configuration addresses Java Card Systems with full capabilities, in particular post-issuance content management; the Remote Method Invocation is optional. Both configurations consider off-card bytecode verification.

The Java Card System - Closed Configuration Protection Profile replaces the Java Card System – Minimal Configuration Protection Profile, version 1.0b, registered under the reference PP/0303.

The Java Card System - Open Configuration Protection Profile replaces the Java Card System – Standard 2.2 Configuration Protection Profile, version 1.0b, registered under the reference PP/0304.

A dedicated Protection Profile shall address the Java Card Platform version 3 Connected Edition.

Table of Contents

Figures

Tables

1 INTRODUCTION

This chapter provides the identification of the Protection Profile, presents its general structure and introduces key notions used in the following chapters.

1.1 PROTECTION PROFILE IDENTIFICATION

Title: Java Card System - Closed Configuration Protection Profile

Version: 3.0

Publication date: December 28th, 2012

Certified by: ANSSI, the French Certification Body

Sponsor: Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065 USA.

Editor: Trusted Labs S.A.S - 5, rue du Bailliage, 78000 Versailles, France.

Review Committee: Java Card Forum – Common Criteria Subgroup

This Protection Profile is conformant to the Common Criteria version 3.1 revision 4.

The minimum assurance level for this Protection Profile is EAL 4 augmented with AVA_VAN.5 "Advanced methodical vulnerability analysis" and ALC_DVS.2 "Sufficiency of security measures".

1.2 PROTECTION PROFILE PRESENTATION

This Protection Profile replaces the Java Card Protection Profile Collection - Standard 2.2 Configuration [PP-JCS-1.0]. It has been developed by Sun Microsystems with the aim of providing a full set of up-to-date security requirements: it relies on current Common Criteria version 3.1 revision 4 and it addresses versions 2.2.x as well as version 3 Classic Edition of the Java Card Specifications, namely [JCVM22], [JCRE22], [JCAPI22], [JCVM221], [JCVM222], [JCAPI221], [JCVM222], [JCRE222], [JCAPI222], [JCVM3], [JCAPI3] and [JCRE3]. Those specifications[1] cover the Java Card platform virtual machine ("Java Card virtual machine" or

[1] In this document, any reference to a specific version of Java Card Platform Specification can be replaced by any newer version of specification. For instance [JCRE22] can be replaced by [JCRE3] but not the inverse.

"Java Card VM"), the Java Card platform runtime environment ("Java Card runtime environment" or "Java Card RE") and the Java Card Application Programming Interface (API).

This Protection Profile applies to evaluations of closed Java Card Platforms, that is, smart cards or similar devices enabled with Java Card technology that does not support post-issuance downloading of applications, referred to as Java Card technology-based applets ("Java Card applets" or "applets"). The Java Card technology combines a subset of the Java programming language with a runtime environment optimized for smart cards and similar small-memory embedded devices. The main security goal of the Java Card platform is to counter the unauthorized disclosure or modification of the code and data (keys, PINs, biometric templates, etc) of applications and platform. In order to achieve this goal, the Java Card System provides strong security features such as the firewall mechanism, dedicated API for security services, etc.

Figure 1 shows the typical architecture of a Java Card Platform (JCP), composed of a Smart Card Platform (SCP), a Java Card System (JCS) and of native code running on top of the SCP. The SCP is the combination of a Security Integrated Circuit (hereafter the "Security IC" or simply the "IC") consisting of processing units, security components, I/O ports and volatile/non-volatile memories, with a native Operating System (hereafter the "OS"). The Java Card System implements the Java Card RE, the Java Card VM and the Java Card API along with the native libraries that supports the Java Card API. The Java Card System provides a layer between the SCP and the applets space. This layer allows applications written for one SCP enabled with Java Card technology to run on any other such platform. The applets space is out of the scope of this Protection Profile.

Native code is usually compiled to the native instruction set of the platform, hence their name. There are two kinds of native code:

- Native Applications: This is native code, which resides in parallel to the Java Card System and can be used from outside the card in the same way as a Java Card applet.
- Native Libraries: This code can be used by the implementation of some Java Card APIs (e.g. cryptographic libraries) or by native applications. This code cannot be used from the outside of the card directly.

Application note:

The "Additional Native Code" block shown in Figure 1 represents all the native code other than the native application implementing the JCS and the native libraries used by the JCS.

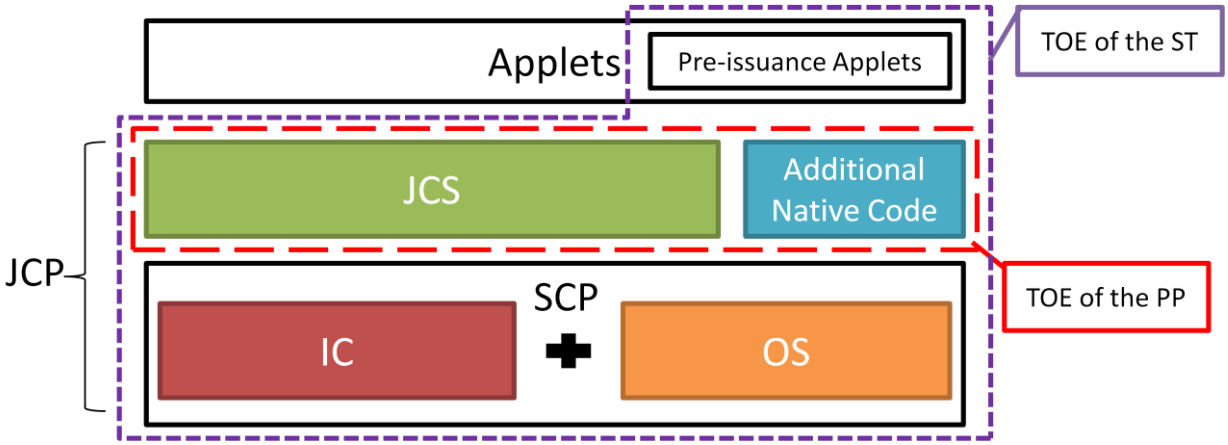

Figure 1: Java Card Platform

This Protection Profile focuses on the security requirements for the JCS and considers the SCP as the environment of the TOE, thus covered by security objectives. Nevertheless, any smart card evaluation against[2] this PP shall comprehend the IC and all the embedded software, including the OS, the JCS, as well as the additional native code and the pre-issuance applets. That is, the TOE of the ST conformant to this PP is as shown in Figure 1. The aim of introducing all the native code in the scope of the evaluation is to test that the native code that does not implement any security function cannot be used to circumvent or jeopardize the JCS TSFs.

This Protection Profile does not require formal compliance to a specific IC Protection Profile or a smart card OS Protection Profile but those IC and OS evaluated against [PP0035] and [PP-ESforSSD] respectively, fully meet the objectives.

This Protection Profile requires "demonstrable" conformance.

The PP has been certified by French Scheme ANSSI.

The structure of this document is as follows:

- Chapter 2 presents an overview of the TOE, its security features and its life cycle.
- Chapter 3 defines the conformance claims applicable to this Protection Profile.
- Chapter 4 introduces general Java Card System security concerns, called "security aspects".
- Chapter 5 presents the assets of the JCS, the links between users and subjects, the relevant threats, the organisational security policies, and the assumptions.
- Chapter 6 describes the TOE security objectives, the security objectives for the operational environment and the security objectives rationale.
- Chapter 7 defines the TOE security functional and assurance requirements, the security requirements rationales, the dependencies analysis and the rationales for assurance requirements.
- Appendix 1 provides a comprehensive view of the two Java Card System Protection Profiles, Open and Closed Configurations.

[2] A product evaluation « against » this PP stands for a product evaluation "claiming conformance to" this PP.

- Appendix 2 contains a glossary of technical terms used in this document.

1.3 REFERENCES

[CC1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model. Version 3.1. Revision 4. September 2012. CCMB-2012-09-001.

[CC2] Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements. Version 3.1. Revision 4. September 2012. CCMB-2012-09-002.

[CC3] Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance requirements. Version 3.1. Revision 4. September 2012. CCMB-2012-09-003.

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology. Version 3.1. Revision 4. September 2012. CCMB-2012-09-004.

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

[GP] GlobalPlatform Card Specification, Version 2.2, March 2006.

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

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

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

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

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

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

[JCVM221] Java Card Platform, version 2.2.1 Virtual Machine (Java Card VM) Specification. October 2003. Published by Sun Microsystems, Inc.

[JCAPI221] Java Card Platform, version 2.2.1 Application Programming Interface.

October 2003. Published by Sun Microsystems, Inc.

[JCRE221] Java Card Platform, version 2.2.1 Runtime Environment (Java Card RE) Specification. October 2003. Published by Sun Microsystems, Inc.

[JCVM222] Java Card Platform, version 2.2.2 Virtual Machine (Java Card VM) Specification. Beta release, October 2005. Published by Sun Microsystems, Inc.

[JCAPI222] Java Card Platform, version 2.2.2 Application Programming Interface, March 2006. Published by Sun Microsystems, Inc.

[JCRE222] Java Card Platform, version 2.2.2 Runtime Environment (Java Card RE) Specification. March 2006. Published by Sun Microsystems, Inc.

[JCVM3] Java Card Platform, versions 3.0 (March 2008) and 3.0.1 (April 2009), Classic Edition, Virtual Machine (Java Card VM) Specification. Published by Sun Microsystems, Inc.

[JCAPI3] Java Card Platform, versions 3.0 (March 2008) and 3.0.1 (April 2009), Classic Edition, Application Programming Interface, March 2008. Published by Sun Microsystems, Inc.

[JCRE3] Java Card Platform, versions 3.0 (March 2008) and 3.0.1 (April 2009), Classic Edition, Runtime Environment (Java Card RE) Specification. March 2008. Published by Sun Microsystems, Inc.

[JCBV] Java Card 3 Platform Off-card Verification Tool Specification, Classic Edition, Version 1.0. Published by Oracle.

[JAVASPEC] The Java Language Specification. Third Edition, May 2005. Gosling, Joy, Steele and Bracha. ISBN 0-321-24678-0.

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

[PP-ESforSSD] Embedded Software for Smart Secure Devices Protection Profile, v1.0, November 27th 2009, ANSSI.

[PP0035] Security IC Platform Protection Profile, Version 1.0, 15 July 2007.

[PP-JCS-1.0] Java Card Protection Profile Collection, Version 1.0b, August 2003, registered and certified by the French certification body (ANSSI) under the following references: [PP/0303] "Minimal Configuration", [PP/0304] "Standard 2.1.1 Configuration", [PP/0305] "Standard 2.2 Configuration" and [PP/0306] "Defensive Configuration".

[PP-JCSO-3.0] Java Card System Protection Profile – Open Configuration, Version 2.6, April 19th 2010, registered and certified by the French certification body (ANSSI) under the reference [ANSSI-CC-PP-2010/03]

2 TOE OVERVIEW

This chapter defines the Target of Evaluation (TOE) type and describes the main security features of the TOE, the components of the TOE environment, the TOE life-cycle and TOE intended usage.

2.1 TOE TYPE

2.1.1 TOE OF THIS PP

The TOE type in this PP is the Java Card System (Java Card RE, Java Card VM and Java Card API) along with the additional native code embedded in a Smart Card Platform. The Java Card System is compliant with Java Card specifications versions 2.2.x or 3 Classic Edition, without post-issuance installation facilities of applications verified off-card. The TOE may implement the Java Card Remote Method Invocation functionality, which is optional in this PP. Native code post-issuance downloading is out of the scope in this PP.

This TOE constitutes the target of the security requirements stated in this Protection Profile. It defines the perimeter of the security requirements stated in this Protection Profile but does not define the perimeter of an actual evaluated product (TOE of the ST) that must include the Smart Card Platform.

2.1.2 TOE OF THE ST

The TOE type of the Security Target (ST) that declares conformity to this PP is the Smart Card Platform (IC and OS) along with the native applications (if any), pre-issuance applets (if any) and the Java Card System.

Any evaluation of a Smart Secure Device against this PP must include this type of TOE. The ST writer shall indicate whether JCRMI is implemented in the TOE and whether it is activated or not. If the TOE provides JCRMI functionality, the full range of SFRs applies. Otherwise, the ST writer shall ignore JCRMI dedicated threats, objectives and requirements.

Application note:

In case where the TOE of the ST includes additional native code that provides security features, the writer of the ST that complies with this PP shall add specific security objectives and security functional requirements for the TOE. He shall then provide full Common Criteria evidence that the additional native code satisfies all these new requirements. In addition, since the TOE of the ST includes the Smart Card Platform which belongs to the operational

environment of the TOE of the PP, all the security objectives on the IC and the OS introduced in this PP shall be redefined as security objectives "on the TOE" in the ST.

2.2 TOE SECURITY FUNCTIONS

The Java Card System Closed Configuration considered in this Protection Profile implements Java Card Specifications versions 2.2.x or 3 Classic Edition without post-issuance downloading of applications.

Figure 2 shows the Java Card System and the relationship with the environment for applet installation purposes in two scenarios: one relying on off-card verification (black lines), described hereafter, the other one relying on on-card verification by the installer (dotted red lines).

The development of the applets is carried on in a Java programming environment. The compilation of the code produces the corresponding class file. Then, this latter file is processed by the converter[3] which validates the code and generates a converted applet (CAP) file, the equivalent of a Java class file for the Java Card platform. A CAP file contains an executable binary representation of the classes of a package. A package is a namespace within the Java programming language that may contain classes and interfaces, and in the context of Java Card technology, it defines either a user library, or one or several applets. Then, the off-card bytecode verifier checks the CAP file (cf. Section 2.3.1 for more details). After the validation is carried out, the CAP file has to be loaded into the card by means of a safe loading mechanism.

The loading of a file into the card embodies two main steps: First an authentication step by which the card issuer and the card recognize each other, for instance by using a type of cryptographic certification. Once the identification step is accomplished, the CAP file is transmitted to the card. Due to resource limitations, usually the file is split by the card issuer into a list of Application Protocol Data Units (APDUs), which are in turn sent to the card. Once loaded into the card the file is linked, what makes it possible in turn to install, if defined, instances of any of the applets defined in the file.

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

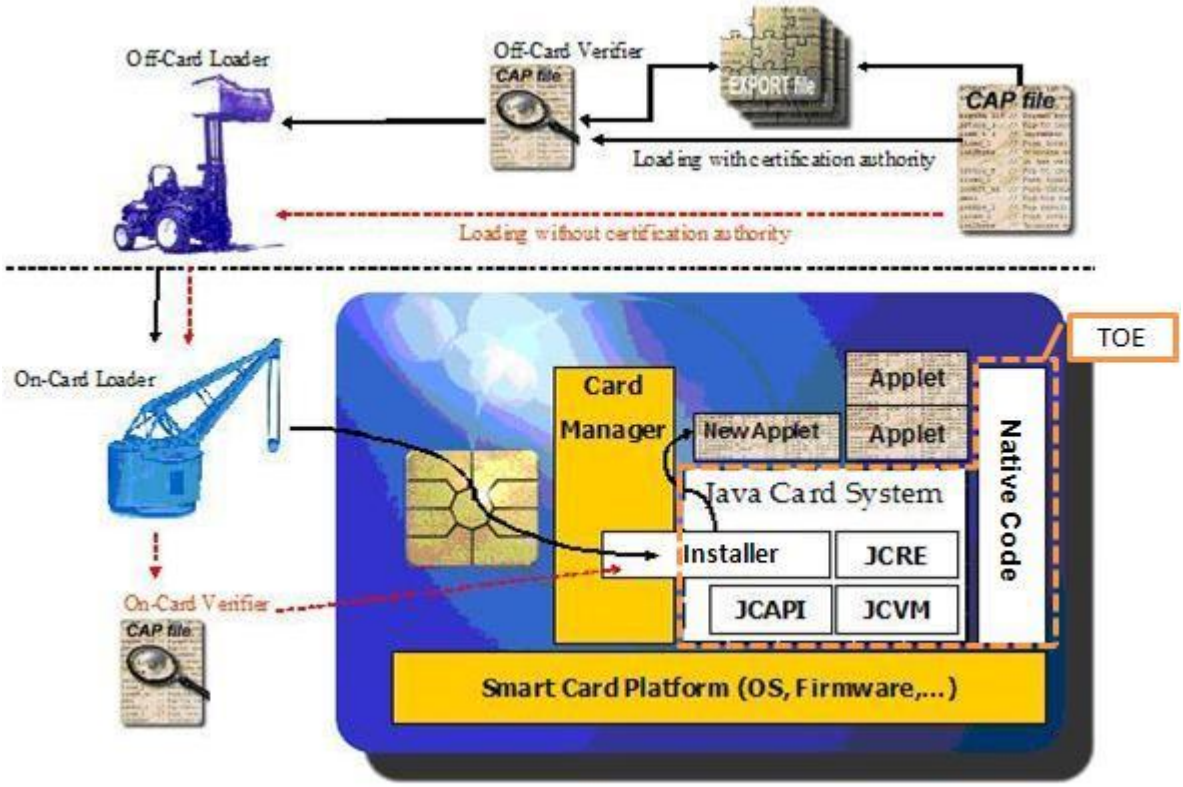

Figure 2: Java Card System and applet installation environment

The linking process consists of a rearrangement of the information contained in the CAP file in order to speed up the execution of the applications. There is a first step where indirect external and internal references contained in the file are resolved by replacing those references with direct ones. This is what is referred to as the resolution step in the [JVM]. In the next step, called the preparation step in [JVM], the static field image[4] and the statically initialized arrays defined in the file are allocated. Those arrays in turn are also initialized, thus giving rise to what shall constitute the initial state of the package for the embedded interpreter.

During the installation process the applet is registered on the card by using an application identifier (AID). This AID will allow the identification of unique applet instances within the card. In particular, the AID is used for selecting the applet instance for execution. In some cases, the actual installation (and registration) of applets is postponed; in the same vein, a package may contain several applets, and some of them might never be installed. Installation is then usually separated from the process of loading and linking a CAP file on the card.

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

The installer is the Java Card System component dealing with loading, linking and installation of new packages, as described in [JCRE22]. Once selected, it receives the CAP file, stores the classes of the package on the card, initializes static data, if any, and installs any applets contained in the package. The installer is also in charge of applet deletion ([JCRE22], §11.3.4):

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

The installer is out of the scope of the TOE. In this Protection Profile, application loading, linking and installation are performed pre-issuance in a controlled environment. The TOE does not provide any content management functionality in the final use operational environment.

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

While the Java Card VM is responsible for ensuring language-level security, the Java Card RE provides additional security features for Java Card technology-enabled devices. Applets from different vendors can coexist in a single card, and they can even share information. An applet, however, is usually intended to store highly sensitive information, so the sharing of that information must be carefully limited. In the Java Card platform, applet isolation is achieved through the applet firewall mechanism ([JCRE22] and [JCRE3] §6.1). That mechanism confines an applet to its own designated memory area, thus each applet is prevented from accessing fields and operations of objects owned by other applets, unless a "shareable interface" is explicitly provided ~~(~~by the applet who owns it~~)~~ for allowing access to that information. The Java Card RE allows sharing using the concept of "shareable interface objects" (SIO) and static public variables. Java Card VM dynamically enforces the firewall, that is, at runtime. However applet isolation cannot be entirely granted by the firewall mechanism if certain integrity conditions are not satisfied by the applications loaded on the card. Those conditions can be statically verified to hold by a bytecode verifier ([JCRE22], §6.1.1).

The Java Card VM ensures that the only way for applets to access any resources are either through the Java Card RE or through the Java Card API (or other vendor-specific APIs). This objective can only be guaranteed if applets are correctly typed (all the "must clauses" imposed in chapter 7 of [JCVM22] on the bytecodes and the correctness of the CAP file format are satisfied).

The Java Card System compliant with Java Card specification versions 2.2.x or 3 Classic Edition supports the Java Card System Remote Method Invocation (JCRMI) and logical channels.

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

Implementation of JCRMI is mandatory in versions 2.2.x of Java Card specification and optional in version 3 Classic Edition. The JCS developer may also choose not to activate this functionality in the TOE. For these reasons, this PP considers that the TOE may provide or not the JCRMI functionality.

Logical channels allow a terminal to open multiple sessions into the smart card, one session per logical channel ([JCRE22], §4). Commands may be issued on a logical channel to instruct the card either to open or to close a logical channel. An applet instance that is selected to be active on a channel shall process all the commands issued to that channel. The platform also introduces the possibility for an applet instance to be selected on multiple logical channels at the same time, or accepting other applets belonging to the same package to be selected simultaneously. These applets are referred to as multiselectable. A non-multiselectable applet can be active at most on one channel. Applets within a package are either all multiselectable or all non-multiselectable.

The Java Card System may optionally provide:

- Object deletion upon request of an applet instance. The JCRE ensures that any unreferenced object owned by that instance is deleted and the associated space is recovered for reuse.
- Extended memory facilities, introduced in Java Card specification version 2.2.2. This is an API-based mechanism to access the external memory outside the addressable Java Card VM space.

Java Card System 2.2.2 also provides support for biometric templates management, external memory access and contactless I/O interface.

Note that the optional features "Extended memory" and "biometric templates" are not part of the TOE since they are not included in the Java Card System 2.2.1. Nonetheless, they are detailed in Appendix 2 for informative purposes.

Lastly, Java Card System 3 Classic Edition provides support for ETSI defined SWP protocol for contactless communication, and for independent contacted and contactless interfaces.
Moreover, it provides support for USB connected interface communication.

2.3 NON-TOE HW/SW/FW AVAILABLE TO THE TOE

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

2.3.1 BYTECODE VERIFICATION

The bytecode verifier is a program that performs static checks on the bytecodes of the methods of a CAP file prior to the execution of the file on the card. Bytecode verification is a key component of security: applet isolation, for instance, depends on the file satisfying the properties a verifier checks to hold. A method of a CAP file that has been verified shall not contain, for instance, an instruction that allows forging a memory address or an instruction that makes improper use of a return address as if it were an object reference. In other words, bytecodes are verified to hold up to the intended use to which they are defined. Bytecode verification could be performed totally or partially dynamically. No standard procedure in that concern has yet been recognized. Furthermore, different approaches have been proposed for the implementation of bytecode verifiers, most notably data flow analysis, model checking and lightweight bytecode verification, this latter being an instance of what is known as proof carrying code. The actual set of checks performed by the verifier is implementation-dependent, but it is required that it should at least enforce all the "must clauses" imposed in [JCVM22] on the bytecodes and the correctness of the CAP files' format.

As for this Protection Profile, the bytecode verifier is an off-card component.

2.3.2 LOADING, LINKING AND INSTALLATION OF APPLETS

The loading, linking and installation of applets in the card occur in a controlled environment prior issuance of the product. The installer in charge of these operations is out of the scope of the TOE.

2.3.3 THE CARD MANAGER (CM)

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

The card manager's role is also to manage and control the communication between the card and the card acceptance device (CAD) or the proximity-coupling device (PCD)[5]. It is the controller of the card, but relies on the TOE to manage the runtime of client applets.
A candidate for this component is the Global Platform card manager [GP].

2.3.4 SMART CARD PLATFORM

The SCP is composed of an IC with a Dedicated Software (DS) if any and a native OS. SCP provides memory management functions, I/O functions that are compliant with ISO standards, transaction facilities and secure (shielded, native) implementation of cryptographic functions. The SCP shall be evaluated along with the TOE in a product evaluation[6].

2.4 TOE LIFE CYCLE

The Java Card System (the TOE) life cycle is part of the product life cycle, i.e. the Java Card platform with applications, which goes from product development to its usage by the final user. The product life cycle phases are those detailed in Figure 3. We refer to [PP0035] for a thorough description of Phases 1 to 7:

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

The Java Card System life cycle is composed of four stages:

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

JCS storage is not necessarily a single step in the life cycle since it can be stored in parts. JCS delivery occurs before storage and may take place more than once if the TOE is delivered in parts. These stages map to the typical smartcard life cycle phases as shown in Figure 3.

[5] The acronym CAD is used here and throughout this specification to refer to both types of card readers - the conventional Card Acceptance Device (CAD) for contacted I/O interfaces and the Proximity Coupling Device (PCD) for contactless interfaces.
[6] This is true for a product which security target is claiming conformance to this PP.

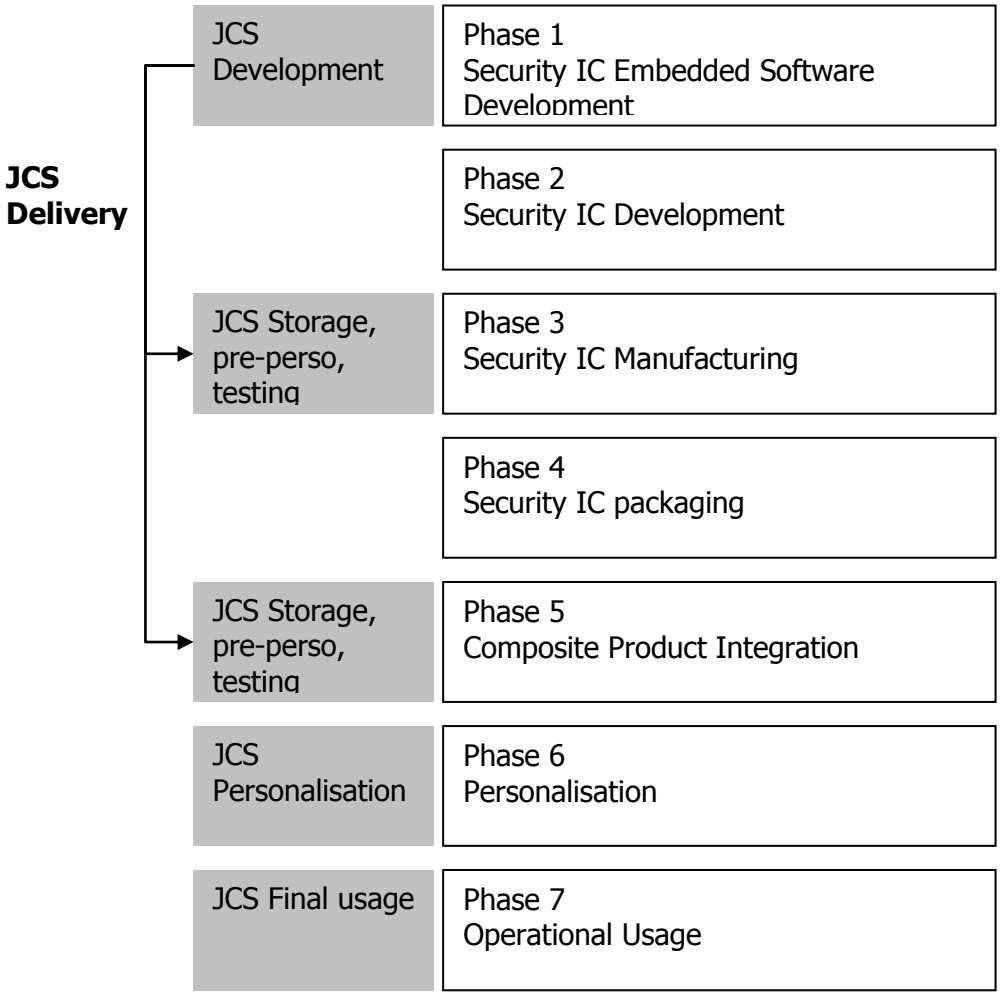

Figure 3: JCS (TOE) Life Cycle within Product Life Cycle

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

The delivery of the JCS may occur either during Security IC Manufacturing (Phase 3) or during Composite Product Integration (Phase 5). It is also possible that part of the JCS is delivered in Phase 3 and the rest is delivered in Phase 5. Delivery and acceptance procedures shall guarantee the authenticity, the confidentiality and integrity of the exchanged pieces. JCS

delivery shall usually involve encrypted signed sending and it supposes the previous exchange of public keys. The evaluation of a product against this PP shall include the delivery process.

In Phase 3, the Security IC Manufacturer may store, pre-personalize the JCS and potentially conduct tests on behalf of the JCS developer. The Security IC Manufacturing environment shall protect the integrity and confidentiality of the JCS and of any related material, for instance test suites. The evaluation of a product against this PP shall include the whole Security IC Manufacturing environment, in particular those locations where the JCS is accessible for installation or testing. If the Security IC has already been certified (e.g. against [PP0035]) there is no need to perform the evaluation again.

In Phase 5, the Composite Product Integrator may store, pre-personalize the JCS and potentially conduct tests on behalf of the JCS developer. The Composite Product Integration environment shall protect the integrity and confidentiality of the JCS and of any related material, for instance test suites. Note that (part of) JCS storage in Phase 5 implies a product delivery after Phase 5. Hence, the evaluation of such product against this PP shall include the Composite Product Integrator environment (may be more than one if there are many integrators). At the end of this stage, all the applets of the final product have been installed on top of the JCS.

The JCS (and potentially the applets) is personalized in Phase 6, if necessary. The Personalization environment shall be included in a product evaluation only if the product delivery point is at the end of Phase 6. This means that some of the product personalization operations may require a controlled environment (secure locations, secure procedures and trusted personnel). The product shall be tested again and all critical material including personalization data, test suites and documentation shall be protected from disclosure and modification.

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

Application note:

The Security Target writer shall specify the life cycle of the product, the JCS delivery point and the product delivery point. The product delivery point may arise at the end of Phase 3, 4, 5 or 6 depending on the product itself. Note that JCS delivery precedes product delivery. During product evaluation against this Protection Profile, the ALC security assurance requirements apply to the whole product life cycle up to delivery.

2.5 TOE USAGE

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

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

Applications installed on a Java Card platform can be selected for execution when the card communicates with a card reader.

Notice that these applications may contain other confidentiality (or integrity) sensitive data than usual cryptographic keys and PINs; for instance, passwords or pass-phrases are as confidential as the PIN, or the balance of an electronic purse.

So far, the most typical applications are:

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

3 CONFORMANCE CLAIMS

3.1 CC CONFORMANCE CLAIMS

This Protection Profile is CC Part 2 [CC2] and CC Part 3 [CC3] conformant of Common Criteria version 3.1, revision 4.

3.2 CONFORMANCE CLAIM TO A PACKAGE

The minimum assurance level for the evaluation of a Java Card Platform with a TOE conformant to this PP is EAL4 augmented with AVA_VAN.5 "Advanced methodical vulnerability analysis" and ALC_DVS.2 "Sufficiency of security measures".

3.3 PROTECTION PROFILE CONFORMANCE CLAIMS

This Protection Profile does not claim conformance to any other Protection Profile.

3.4 CONFORMANCE CLAIMS TO THIS PROTECTION PROFILE

The conformance to this PP, required for the Security Targets and Protection Profiles claiming conformance to it, is demonstrable, as defined in CC Part 1 [CC1].

4 SECURITY ASPECTS

This chapter describes the main security issues of the Java Card System and its environment addressed in this Protection Profile, called "security aspects", in a CC-independent way. In addition to this, they also give a semi-formal framework to express the CC security environment and objectives of the TOE. They can be instantiated as assumptions, threats, objectives (for the TOE and the environment) or organizational security policies. For instance, we will define hereafter the following aspect:

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

TSFs must be continuously active in one way or another; this is called "OPERATE". The Protection Profile may include an assumption, called "A.OPERATE", stating that it is assumed that the TOE ensures continued correct operation of its security functions, and so on. However, it may also include a threat, called "T.OPERATE", to be interpreted as the negation of the statement #.OPERATE. In this example, this amounts to stating that an attacker may try to circumvent some specific TSF by temporarily shutting it down. The use of "OPERATE" is intended to ease the understanding of this document.

This section presents security aspects that will be used in the remainder of this document. Some being quite general, we give further details, which are numbered for easier 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.

4.1 CONFIDENTIALITY

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

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

#.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 RE, the Java Card VM and the internal data of Java Card platform API classes as well.

4.2 INTEGRITY

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

#.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.

#.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 RE, the Java Card VM and the internal data of Java Card API classes as well.

4.3 UNAUTHORIZED EXECUTIONS

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

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

#.FIREWALL The Firewall shall ensure controlled sharing of class instances[7], and isolation of their data and code between packages (that is, controlled execution contexts) as well as between packages and the JCRE context. An applet shall not read, write, compare a piece of data belonging to an applet that is not in the same context, or execute one of the methods of an applet in another context without its authorization.

#.NATIVE Because the execution of native code is outside of the JCS TSF scope, it must be secured so as to not provide ways to bypass the TSFs of the JCS. Loading of native code, which is as well outside those TSFs, is submitted to the same requirements. Should native software be privileged in this respect, exceptions to the policies must include a rationale for the new security framework they introduce.

4.4 BYTECODE VERIFICATION

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

4.4.1 CAP FILE VERIFICATION

Bytecode verification includes checking at least the following properties: (3) bytecode instructions represent a legal set of instructions used on the Java Card platform; (4) adequacy of bytecode operands to bytecode semantics; (5) absence of operand stack overflow/underflow; (6) control flow confinement to the current method (that is, no control jumps to outside the method); (7) absence of illegal data conversion and reference forging; (8) enforcement of the private/public access modifiers for class and class members; (9) validity of any kind of reference used in the bytecodes (that is, any pointer to a bytecode, class, method, object, local variable, etc actually points to the beginning of piece of data of the expected kind); (10) enforcement of rules for binary compatibility (full details are given in [JCVM22], [JVM], [JCBV]). The actual set of checks performed by the verifier is implementation-dependent, but shall at least enforce all the "must clauses" imposed in [JCVM22] on the bytecodes and the correctness of the CAP files' format.

As most of the actual Java Card VMs do not perform all the required checks at runtime, mainly because smart cards lack memory and CPU resources, CAP file verification prior to execution is mandatory. On the other hand, there is no requirement on the precise moment when the verification shall actually take place, as far as it can be ensured that the verified file is not

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

modified thereafter. Therefore, the bytecodes can be verified either before the loading of the file on to the card or before the installation of the file in the card or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time. This Protection Profile assumes bytecode verification is performed off-card.

Another important aspect to be considered about bytecode verification and application downloading is, first, the assurance that every package required by the loaded applet is indeed on the card, in a binary-compatible version (binary compatibility is explained in [JCVM22] §4.4), second, that the export files used to check and link the loaded applet have the corresponding correct counterpart on the card.

4.4.2 INTEGRITY AND AUTHENTICATION

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

Alternatively, the card itself may include a verifier and perform the checks prior to the effective installation of the applet or provide means for the bytecodes to be verified dynamically. On-card bytecode verifier is out of the scope of this Protection Profile.

4.4.3 LINKING AND VERIFICATION

Beyond functional issues, the installer ensures at least a property that matters for security: the loading order shall guarantee that each newly loaded package references only 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.

4.5 CARD MANAGEMENT

#.CARD-MANAGEMENT The card manager shall implement the card issuer's policy on the card.

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

#.SID (1) Users and subjects of the TOE must be identified. (2) The identity of sensitive users and subjects associated with administrative and privileged roles must be particularly protected; this concerns the Java Card RE, the applets registered on the card, and especially the default applet and the currently selected applet (and all other active applets in Java Card System 2.2.x). A change of identity, especially standing for an administrative role (like an applet impersonating the Java Card RE), is a severe violation of the Security Functional Requirements (SFR). Selection controls the access to any data exchange between the TOE and the CAD and therefore, must be protected as well. 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 (1) Deallocation of objects should not introduce security holes in the form of references pointing to memory zones that are not longer in use, or have been reused for other purposes. Deletion of collection of objects should not be maliciously used to circumvent the TSFs. (2) Erasure, if deemed successful, shall ensure that the deleted class instance is no longer accessible.

4.6 SERVICES

#.ALARM The TOE shall provide appropriate feedback upon detection of a potential security violation. This particularly concerns the type errors detected by the bytecode verifier, the security exceptions thrown by the Java Card VM, 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 SFRs. Then: (1) After a power loss, RF signal loss or sudden card removal prior to completion of some communication protocol, the SCP will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state. (2) It does not allow the SFRs to be bypassed or altered and does not allow access to other low-level functions than those made available by the packages of the Java Card API. That includes the protection of its private data and code (against disclosure or modification) from the Java Card System. (3) It provides secure low-level cryptographic processing to the Java Card System. (4) It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism. (5) It allows the Java Card System to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low–level control accesses (segmentation fault detection). (6) It safely transmits low–level exceptions to the TOE (arithmetic exceptions, checksum errors), when applicable. Finally, it is required that (7) the IC is designed in accordance with a well-defined set of policies and standards (for instance, those specified in [PP0035]), and will be tamper resistant to actually prevent an attacker from extracting or altering security data (like cryptographic keys) by using commonly employed techniques (physical probing and sophisticated analysis of the chip). This especially matters to the management (storage and operation) of cryptographic keys.

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

5 SECURITY PROBLEM DEFINITION

5.1 ASSETS

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

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

The assets to be protected by the TOE are listed below. They are grouped according to whether it is data created by and for the user (User data) or data created by and for the TOE (TSF data). For each asset it is specified the kind of dangers that weigh on it.

5.1.1 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.APP_KEYs

Cryptographic keys owned by the applets.

To be protected from unauthorized disclosure and modification.

D.PIN

Any end-user's PIN.

To be protected from unauthorized disclosure and modification.

5.1.2 TSF DATA

D.API_DATA

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

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.

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 Java Card VM, such as, for instance, the frame stack, the program counter, the class of an object, the length allocated for an array, any pointer used to chain data-structures.

To be protected from unauthorized disclosure or modification.

D.SEC_DATA

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

To be protected from unauthorized disclosure and modification.

5.2 THREATS

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

5.2.1 CONFIDENTIALITY

T.CONFID-APPLI-DATA

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

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

T.CONFID-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.CONFID-JCS-DATA

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

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

5.2.2 INTEGRITY

T.INTEG-APPLI-CODE

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-CODE.LOAD

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-DATA

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

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

T.INTEG-APPLI-DATA.LOAD

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

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

T.INTEG-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.INTEG-JCS-DATA

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

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

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

5.2.3 IDENTITY USURPATION

T.SID.1

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

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

T.SID.2

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

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

5.2.4 UNAUTHORIZED EXECUTION

T.EXE-CODE.1

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

Directly threatened asset(s): D.APP_CODE.

T.EXE-CODE.2

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

Directly threatened asset(s): D.APP_CODE.

T.EXE-CODE-REMOTE

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

Directly threatened asset(s): D.APP_CODE.

Application Note:

This threat concerns versions 2.2.x and 3 Classic Edition of the Java Card RMI, which allow external users (that is, other than on-card applets) to trigger the execution of code belonging to an on-card applet. On the contrary, T.EXE-CODE.1 is restricted to the applets under the TSF.

This threat applies only if the TOE provides JCRMI functionality.

T.NATIVE

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

Directly threatened asset(s): D.JCS_DATA.

5.2.5 DENIAL OF SERVICE

T.RESOURCES

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

Directly threatened asset(s): D.JCS_DATA.

5.2.6 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 for further details.

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

5.2.7 MISCELLANEOUS

T.PHYSICAL

The attacker discloses or modifies the design of the TOE, its sensitive data or application code by physical (opposed to logical) tampering means. This threat includes IC failure analysis, electrical probing, unexpected tearing, and DPA. That also includes the modification of the runtime execution of Java Card System or SCP software through alteration of the intended execution order of (set of) instructions through physical tampering techniques.

This threatens all the identified assets.

This threat refers to the point (7) of the security aspect #.SCP, and all aspects related to confidentiality and integrity of code and data.

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

If the application development guidance provided by the platform developer contains recommendations related to the isolation property of the platform, this policy shall also ensure that the verification authority checks that these recommendations are applied in the application code.

5.4 ASSUMPTIONS

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

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.NO-DELETION

No deletion of installed applets (or packages) is possible.

A.NO-INSTALL

There is no post-issuance installation of applets. Installation of applets is secure and occurs only in a controlled environment in the pre-issuance phase. See #.INSTALL for details.

6 SECURITY OBJECTIVES

6.1 SECURITY OBJECTIVES FOR THE TOE

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

6.1.1 IDENTIFICATION

O.SID

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

6.1.2 EXECUTION

O.FIREWALL

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

O.GLOBAL_ARRAYS_CONFID

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

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

O.GLOBAL_ARRAYS_INTEG

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

O.NATIVE

The only means that the Java Card VM shall provide for an application to execute native code is the invocation of a method of the Java Card API, or any additional API. See #.NATIVE for details.

O.OPERATE

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

O.REALLOCATION

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

O.RESOURCES

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

6.1.3 SERVICES

O.ALARM

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

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.

O.PIN-MNGT

The TOE shall provide a means to securely manage PIN objects. See #.PIN-MNGT 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.REMOTE

The TOE shall provide restricted remote access from the CAD to the services implemented by the applets on the card. This particularly concerns the Java Card RMI services introduced in version 2.2.x of the Java Card platform and that became optional in version 3 Classic Edition.

Application Note:

This objective applies only if the TOE provides JCRMI functionality.

O.TRANSACTION

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

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. These proprietary libraries will be evaluated together with the TOE.

6.1.4 OBJECT DELETION

O.OBJ-DELETION

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

6.2 SECURITY OBJECTIVES FOR THE OPERATIONAL ENVIRONMENT

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

OE.CARD-MANAGEMENT

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

The card manager is an application with specific rights, which is responsible for the administration of the smart card. This component will in practice be tightly connected with the TOE, which in turn shall very likely rely on the card manager for the effective enforcing of some of its security functions. Typically the card manager shall be in charge of the life cycle of the whole card, as well as that of the installed applications (applets). The card manager should prevent that card content management (loading, installation, deletion) is carried out, for instance, at invalid states of the card or by non-authorized actors. It shall also enforce security policies established by the card issuer.

OE.NO-DELETION

No installed applets (or packages) shall be deleted from the card.

OE.NO-INSTALL

There is no post-issuance installation of applets. Installation of applets is secure and shall occur only in a controlled environment in the pre-issuance phase.

OE.SCP.IC

The SCP shall provide all IC security features against physical attacks.

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

- It is required that the IC is designed in accordance with a well-defined set of policies and Standards (likely specified in another protection profile), and will be tamper resistant to actually prevent an attacker from extracting or altering security data (like cryptographic keys) by using commonly employed techniques (physical probing and sophisticated analysis of the chip). This especially matters to the management (storage and operation) of cryptographic keys.

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.

This security objective for the environment refers to the security aspect #.SCP(1): The smart card platform must be secure with respect to the SFRs. Then after a power loss or sudden card removal prior to completion of some communication protocol, the SCP will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state.

OE.SCP.SUPPORT

The SCP shall support the TSFs of the TOE.

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

(2) It does not allow the TSFs to be bypassed or altered and does not allow access to other low-level functions than those made available by the packages of the API. That includes the protection of its private data and code (against disclosure or modification) from the Java Card System.

(3) It provides secure low-level cryptographic processing to the Java Card System.

(4) It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism.

(5) It allows the Java Card System to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low-level control accesses (segmentation fault detection).

OE.VERIFICATION

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

Additionally, the applet shall follow all the recommendations, if any, mandated in the platform guidance for maintaining the isolation property of the platform.

Application Note:

Constraints to maintain the isolation property of the platform are provided by the platform developer in application development guidance. The constraints apply to all application code loaded in the platform.

OE.CODE-EVIDENCE

For application code loaded pre-issuance, evaluated technical measures implemented by the TOE or audited organizational measures must ensure that loaded application has not been changed since the code verifications required in OE.VERIFICATION.

The objectives OE.NO-INSTALL and OE.NO-DELETION have been included so as to describe procedures that shall contribute to ensure that the TOE will be used in a secure manner. Moreover, they have been defined in accordance with the environmental assumptions they

uphold (actually, they are just a reformulation of the corresponding assumptions). The NO-DELETION and NO-INSTALL (assumptions and objectives) constitute the explicit statement that the Closed configuration corresponds to that of a closed card (no code can be loaded or deleted once the card has been issued).

6.3 SECURITY OBJECTIVES RATIONALE

6.3.1 THREATS

6.3.1.1 CONFIDENTIALITY

T.CONFID-APPLI-DATA This threat is countered by the security objective for the operational environment regarding bytecode verification (OE.VERIFICATION). It is also covered by the isolation commitments stated in the (O.FIREWALL) objective. It 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.

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

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

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, PIN's are particular cases of an application's sensitive data (the Java Card System may possess keys as well) 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) shall contribute in covering this threat by controlling the sharing of the global PIN between the applets.

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 security objective O.GLOBAL_ARRAYS_CONFID.

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.

T.CONFID-JCS-CODE This threat is countered by the list of properties described in the (#.VERIFICATION) security aspect. 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 accessibility. As none of those instructions enables reading a piece of code, no Java Card applet can therefore be executed to disclose a piece of code. Native

applications are also harmless because of the objective O.NATIVE, so no application can be run to disclose a piece of code.

The (#.VERIFICATION) security aspect is addressed in this PP by the objective for the environment OE.VERIFICATION.

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

T.CONFID-JCS-DATA This threat is covered by bytecode verification (OE.VERIFICATION) and the isolation commitments stated in the (O.FIREWALL) security 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.

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

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

6.3.1.2 INTEGRITY

T.INTEG-APPLI-CODE This threat is countered by the list of properties described in the (#.VERIFICATION) security aspect. 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 accessibility. As none of these instructions enables modifying a piece of code, no Java Card applet can therefore be executed to modify a piece of code. Native applications are also harmless because of the objective O.NATIVE, so no application can run to modify a piece of code.

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

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that integrity and authenticity evidences exist for the application code loaded into the platform.

T.INTEG-APPLI-CODE.LOAD The objective OE.CODE-EVIDENCE covers this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity. By controlling the access to card management functions such as the installation, update or deletion of applets the objective OE.CARD-MANAGEMENT contributes to cover this threat.

T.INTEG-APPLI-DATA This threat is countered by bytecode verification (OE.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.

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity.

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

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 (the Java Card System may possess keys as well) 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 integrity of the information stored in that buffer is ensured by the objective O.GLOBAL_ARRAYS_INTEG.

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.

T.INTEG-APPLI-DATA.LOAD The objective OE.CODE-EVIDENCE covers this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity. By controlling the access to card management functions such as the installation, update or deletion of applets the objective OE.CARD-MANAGEMENT contributes to cover this threat.

T.INTEG-JCS-CODE This threat is countered by the list of properties described in the (#.VERIFICATION) security aspect. 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 accessibility. As none of these instructions enables modifying a piece of code, no Java Card applet can therefore be executed to modify a piece of code. Native applications are also harmless because of the objective O.NATIVE, so no application can be run to modify a piece of code.

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

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity.

T.INTEG-JCS-DATA This threat is countered by bytecode verification (OE.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.

The objectives OE.CARD-MANAGEMENT and OE.VERIFICATION contribute to cover this threat by controlling the access to card management functions and by checking the bytecode, respectively.

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity.

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

6.3.1.3 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. It should be noticed that the AIDs, which are used for applet identification, are TSF data.

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 objectives O.GLOBAL_ARRAYS_CONFID and O.GLOBAL_ARRAYS_INTEG.

The objective OE.CARD-MANAGEMENT contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter this threat.

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 objectives OE.SCP.RECOVERY and OE.SCP.SUPPORT are intended to support the O.OPERATE objective of the TOE, so they are indirectly related to the threats that this latter objective contributes to counter.

6.3.1.4 UNAUTHORIZED EXECUTION

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 aspect #.VERIFICATION (access modifiers and scope of accessibility 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 aspect related to control flow confinement and the validity of the method references used in the bytecodes.

T.EXE-CODE-REMOTE If the TOE provides JCRMI functionality, the O.REMOTE security objective contributes to prevent the invocation of a method that is not supposed to be accessible from outside the card.

T.NATIVE This threat is countered by O.NATIVE which ensures that a Java Card applet can only access native methods indirectly that is, through an API. 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).

6.3.1.5 DENIAL OF SERVICE

T.RESOURCES This threat is directly countered by objectives on resource-management (O.RESOURCES) for runtime purposes and good working order (O.OPERATE) in a general manner.

It should be noticed 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 Protection Profile, though.

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

6.3.1.6 SERVICES

T.OBJ-DELETION This threat is covered by the O.OBJ-DELETION security objective which ensures that object deletion shall not break references to objects.

6.3.1.7 MISCELLANEOUS

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

6.3.2 ORGANISATIONAL SECURITY POLICIES

OSP.VERIFICATION This policy is upheld by the security objective of the environment OE.VERIFICATION which guarantees that all the bytecodes shall be verified at least once, before the loading, before the installation or before the execution in order to ensure that each bytecode is valid at execution time.

This policy is also upheld by the security objective of the environment OE.CODE-EVIDENCE which ensures that evidences exist that the application code has been verified and not changed after verification.

6.3.3 ASSUMPTIONS

A.VERIFICATION This assumption is upheld by the security objective on the operational environment OE.VERIFICATION which guarantees that all the bytecodes shall be verified at least once, before the loading, before the installation or before the execution in order to ensure that each bytecode is valid at execution time.

This assumption is also upheld by the security objective of the environment OE.CODE-EVIDENCE which ensures that evidences exist that the application code has been verified and not changed after verification.

A.NO-DELETION The assumption A.NO-DELETION is upheld by the environmental objective OE.NO-DELETION which guarantees that no installed applets (or packages) shall be deleted from the card. The environmental objective OE.CARD-MANAGEMENT also upholds this assumption by controlling the access to card management functions such as applets deletion.

A.NO-INSTALL This assumption is upheld by the environmental objective OE.NO-INSTALL which imposes that no post-issuance installation of applets is permitted. The environmental objective OE.CARD-MANAGEMENT contributes in upholding this assumption by controlling the access to card management functions such as the installation of applets.

6.3.4 SPD AND SECURITY OBJECTIVES

Threats	Security Objectives	Rationale
T.CONFID-APPLI-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARD-MANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.ALARM, O.TRANSACTION, O.CIPHER, O.PIN-MNGT, O.KEY-MNGT, O.REALLOCATION	Section 6.3.1
T.CONFID-JCS-CODE	OE.VERIFICATION, OE.CARD-MANAGEMENT, O.NATIVE	Section 6.3.1
T.CONFID-JCS-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARD-MANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.ALARM	Section 6.3.1
T.INTEG-APPLI-CODE	OE.CARD-MANAGEMENT, OE.VERIFICATION, O.NATIVE, OE.CODE-EVIDENCE	Section 6.3.1
T.INTEG-APPLI-CODE.LOAD	OE.CARD-MANAGEMENT, OE.CODE-EVIDENCE	Section 6.3.1
T.INTEG-APPLI-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARD-MANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.GLOBAL_ARRAYS_INTEG, O.ALARM, O.TRANSACTION, O.CIPHER, O.PIN-MNGT, O.KEY-MNGT, O.REALLOCATION, OE.CODE-EVIDENCE	Section 6.3.1
T.INTEG-APPLI-DATA.LOAD	OE.CARD-MANAGEMENT, OE.CODE-EVIDENCE	Section 6.3.1
T.INTEG-JCS-CODE	OE.CARD-MANAGEMENT, OE.VERIFICATION, O.NATIVE, OE.CODE-EVIDENCE	Section 6.3.1
T.INTEG-JCS-DATA	OE.SCP.RECOVERY, OE.SCP.SUPPORT, OE.CARD-MANAGEMENT, OE.VERIFICATION, O.SID, O.OPERATE, O.FIREWALL, O.ALARM, OE.CODE-EVIDENCE	Section 6.3.1
T.SID.1	OE.CARD-MANAGEMENT, O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.GLOBAL_ARRAYS_INTEG, O.SID	Section 6.3.1
T.SID.2	OE.SCP.RECOVERY, OE.SCP.SUPPORT, O.SID,	Section

	O.OPERATE, O.FIREWALL	6.3.1
T.EXE-CODE.1	OE.VERIFICATION, O.FIREWALL	Section 6.3.1
T.EXE-CODE.2	OE.VERIFICATION	Section 6.3.1
T.EXE-CODE-REMOTE	O.REMOTE	Section 6.3.1
T.NATIVE	OE.VERIFICATION, O.NATIVE	Section 6.3.1
T.RESOURCES	O.OPERATE, O.RESOURCES, OE.SCP.RECOVERY, OE.SCP.SUPPORT	Section 6.3.1
T.OBJ-DELETION	O.OBJ-DELETION	Section 6.3.1
T.PHYSICAL	OE.SCP.IC	Section 6.3.1

Table 1 Threats and Security Objectives - Coverage

Security Objectives	Threats
O.SID	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.1, T.SID.2
O.FIREWALL	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.1, T.SID.2, T.EXE-CODE.1
O.GLOBAL_ARRAYS_CONFID	T.CONFID-APPLI-DATA, T.SID.1
O.GLOBAL_ARRAYS_INTEG	T.INTEG-APPLI-DATA, T.SID.1
O.NATIVE	T.CONFID-JCS-CODE, T.INTEG-APPLI-CODE, T.INTEG-JCS-CODE, T.NATIVE
O.OPERATE	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
O.REALLOCATION	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.RESOURCES	T.RESOURCES
O.ALARM	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA
O.CIPHER	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.KEY-MNGT	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.PIN-MNGT	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.REMOTE	T.EXE-CODE-REMOTE
O.TRANSACTION	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.OBJ-DELETION	T.OBJ-DELETION
OE.CARD-MANAGEMENT	T.CONFID-APPLI-DATA, T.CONFID-JCS-CODE, T.CONFID-JCS-DATA, T.INTEG-APPLI-CODE, T.INTEG-APPLI-CODE.LOAD, T.INTEG-APPLI-DATA, T.INTEG-APPLI-DATA.LOAD, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA, T.SID.1
OE.NO-DELETION	
OE.NO-INSTALL	
OE.SCP.IC	T.PHYSICAL
OE.SCP.RECOVERY	T.CONFID-APPLI-DATA, T.CONFID-JCS-

	DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
OE.SCP.SUPPORT	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
OE.VERIFICATION	T.CONFID-APPLI-DATA, T.CONFID-JCS-CODE, T.CONFID-JCS-DATA, T.INTEG-APPLI-CODE, T.INTEG-APPLI-DATA, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA, T.EXE-CODE.1, T.EXE-CODE.2, T.NATIVE
OE.CODE-EVIDENCE	T.INTEG-APPLI-CODE, T.INTEG-APPLI-CODE.LOAD, T.INTEG-APPLI-DATA, T.INTEG-APPLI-DATA.LOAD, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA

Table 2 Security Objectives and Threats - Coverage

Organisational Security Policies	Security Objectives	Rationale
OSP.VERIFICATION	OE.VERIFICATION, OE.CODE-EVIDENCE	Section 6.3.2

Table 3 OSPs and Security Objectives - Coverage

Security Objectives	Organisational Security Policies
O.SID	
O.FIREWALL	
O.GLOBAL_ARRAYS_CONFID	
O.GLOBAL_ARRAYS_INTEG	
O.NATIVE	
O.OPERATE	
O.REALLOCATION	
O.RESOURCES	
O.ALARM	
O.CIPHER	
O.KEY-MNGT	
O.PIN-MNGT	
O.REMOTE	
O.TRANSACTION	
O.OBJ-DELETION	
OE.CARD-MANAGEMENT	
OE.NO-DELETION	
OE.NO-INSTALL	
OE.SCP.IC	
OE.SCP.RECOVERY	
OE.SCP.SUPPORT	
OE.VERIFICATION	OSP.VERIFICATION
OE.CODE-EVIDENCE	OSP.VERIFICATION

Table 4 Security Objectives and OSPs - Coverage

Assumptions	Security Objectives for the Operational Environment	Rationale
A.VERIFICATION	OE.VERIFICATION, OE.CODE-EVIDENCE	Section 6.3.3
A.NO-DELETION	OE.CARD-MANAGEMENT, OE.NO-DELETION	Section 6.3.3
A.NO-INSTALL	OE.CARD-MANAGEMENT, OE.NO-INSTALL	Section 6.3.3

Table 5 Assumptions and Security Objectives for the Operational Environment - Coverage

Security Objectives for the Operational Environment	Assumptions
OE.CARD-MANAGEMENT	A.NO-DELETION, A.NO-INSTALL
OE.NO-DELETION	A.NO-DELETION
OE.NO-INSTALL	A.NO-INSTALL
OE.SCP.IC	
OE.SCP.RECOVERY	
OE.SCP.SUPPORT	
OE.VERIFICATION	A.VERIFICATION
OE.CODE-EVIDENCE	A.VERIFICATION

Table 6 Security Objectives for the Operational Environment and Assumptions - Coverage

7 SECURITY REQUIREMENTS

7.1 SECURITY FUNCTIONAL REQUIREMENTS

This section states the security functional requirements for the Java Card System - Closed configuration. For readability and for compatibility with the original Java Card System Protection Profile Collection - Standard 2.2 Configuration [PP/0305], requirements are arranged into groups. All the groups defined in the table below apply to this Protection Profile.

The SFRs refer to all potentially applicable subjects, objects, information, operations and security attributes, including JCRMI related entities which are optional. If the TOE does not provide JCRMI functionality, the ST writer shall ignore such entities and their corresponding requirements.

Group	Description
Core with Logical Channels (*CoreG_LC*)	The CoreG_LC contains the requirements concerning the runtime environment of the Java Card System implementing logical channels. This includes the firewall policy and the requirements related to the Java Card API. Logical channels are a Java Card specification version 2.2 feature. This group is the union of requirements from the Core (*CoreG*) and the Logical channels (*LCG*) groups defined in [PP/0305] (cf. Java Card System Protection Profile Collection [PP-JCS-1.0]).
Remote Method Invocation (RMI)	The RMIG contains the security requirements for the remote method invocation feature, which provides a new protocol of communication between the terminal and the applets. This feature was introduced in Java Card specification version 2.2 and became optional in Java Card specification version 3 Classic Edition. This group of SFRs applies only if the TOE provides JCRMI functionality.
Object deletion (*ODELG*)	The ODELG contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card specification version 2.2 feature.

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

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

Subject	Description
S.CAD	If the TOE provides JCRMI functionality, this subject can also plays the role of the actor that requests, by issuing commands to the card, for RMI services.
S.JCRE	The runtime environment under which Java programs in a smart card are executed.
S.JCVM	The bytecode interpreter that enforces the firewall at runtime.
S.LOCAL	Operand stack of a JCVM frame, or local variable of a JCVM frame containing an object or an array of references.
S.MEMBER	Any object's field, static field or array position.
S.PACKAGE	A package is a namespace within the Java programming language that may contain classes and interfaces, and in the context of Java Card technology, it defines either a user library, or one or several applets.

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

Object	Description
O.APPLET	Any installed applet, its code and data.
O.JAVAOBJECT	Java class instance or array. It should be noticed that KEYS, PIN, arrays and applet instances are specific objects in the Java programming language.
O.REMOTE_MTHD	A method of a remote interface. It applies only if the TOE provides JCRMI functionality.
O.REMOTE_OBJ	A remote object is an instance of a class that implements one (or more) remote interfaces. The remote interface can extend, directly or indirectly, the interface java.rmi.Remote ([JCAPI22]). It applies only if the TOE provides JCRMI functionality.
O.RMI_SERVICE	These are instances of the class javacardx.rmi.RMIService. They are the objects that actually process the RMI services. It applies only if the TOE provides JCRMI functionality.
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. It applies only if the TOE provides JCRMI functionality.

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

Information	Description
I.DATA	JCVM Reference Data: objectref addresses of APDU buffer, JCRE-owned instances of APDU class and byte array for install method.
I.RORD	Remote object reference descriptors which provide information concerning: (i) the identification of the remote object and (ii) the implementation class of the object or the interfaces implemented by the class of the object. The descriptor is the only object's information to which the CAD can access. It applies only if the TOE provides JCRMI functionality.

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

Security attribute	Description/Value
Active Applets	The set of the active applets' AIDs. An active applet is an applet that is selected on at least one of the logical channels.
Applet Selection Status	"Selected" or "Deselected".
Applet's version number	The version number of an applet (package) indicated in the export file.
Class	Identifies the implementation class of the remote object. It applies only if the TOE provides JCRMI functionality.
Context	Package AID or "Java Card RE".
Currently Active Context	Package AID or "Java Card RE".
ExportedInfo	Boolean (indicates whether the remote object is exportable or not).
Identifier	The Identifier of a remote object or method is a number that uniquely identifies the remote object or method, respectively. It applies only if the TOE provides JCRMI functionality.
LC Selection Status	Multiselectable, Non-multiselectable or "None".
LifeTime	CLEAR_ON_DESELECT or PERSISTENT (*).
Owner	The Owner of an object is either the applet instance that created the object or the package (library) where it has been defined (these latter objects can only be arrays that initialize static fields of the package). If the TOE provides JCRMI functionality, the owner of a remote object is the applet instance that created the object.
Package AID	The AID of each package indicated in the export file.
Registered Applets	The set of AID of the applet instances registered on the card.
Remote	An object is Remote if it is an instance of a class that directly or indirectly implements the interface java.rmi.Remote. It applies only if the TOE provides JCRMI functionality.
Returned References	The set of remote object references that have been sent to the CAD during the applet selection session. This attribute is implementation dependent. It applies only if the TOE provides JCRMI functionality.
Selected Applet Context	Package AID or "None".
Sharing	Standards, SIO, Java Card RE entry point or global array.

(*) Transient objects of type CLEAR_ON_RESET behave like persistent objects in that they can be accessed only when the Currently Active Context is the object's context.

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

Operation	Description
OP.ARRAY_ACCESS(O.JAVAOBJECT, field)	Read/Write an array component.
OP.CREATE(Sharing, LifeTime) (*)	Creation of an object (new or makeTransient call).
OP.GET_ROR(O.APPLET,...)	This operation 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. It applies only if the TOE provides JCRMI functionality.
OP.INSTANCE_FIELD(O.JAVAOBJECT, field)	Read/Write a field of an instance of a class in the Java programming language.
OP.INVK_VIRTUAL(O.JAVAOBJECT, method, arg1,...)	Invoke a virtual method (either on a class instance or an array object).
OP.INVK_INTERFACE(O.JAVAOBJECT, method, arg1,...)	Invoke an interface method.
OP.INVOKE(O.RMI_SERVICE,...)	This operation requests a remote method invocation on the remote object. It applies only if the TOE provides JCRMI functionality.
OP.JAVA(...)	Any access in the sense of [JCRE22], §6.2.8. It stands for one of the operations OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW, OP.TYPE_ACCESS.
OP.PUT(S1,S2,I)	Transfer a piece of information I from S1 to S2.
OP.RET_RORD(S.JCRE,S.CAD,I.RORD)	Send a remote object reference descriptor to the CAD. It applies only if the TOE provides JCRMI functionality.
OP.THROW(O.JAVAOBJECT)	Throwing of an object (athrow, see [JCRE22], §6.2.8.7).
OP.TYPE_ACCESS(O.JAVAOBJECT, class)	Invoke checkcast or instanceof on an object in order to access to classes (standard or shareable interfaces objects).

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

7.1.1 CoreG_LC Security Functional Requirements

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

7.1.1.1 Firewall Policy

FDP_ACC.2/FIREWALL Complete access control

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

Refinement:

The operations involved in the policy are:

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

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

Application Note:

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

FDP_ACF.1/FIREWALL Security attribute based access control

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

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

.

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

- **R.JAVA.1 ([JCRE22], §6.2.8): S.PACKAGE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW or OP.TYPE_ACCESS upon any O.JAVAOBJECT whose Sharing attribute has value "JCRE entry point" or "global array".**
- **R.JAVA.2 ([JCRE22], §6.2.8): S.PACKAGE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE or OP.THROW upon any O.JAVAOBJECT whose Sharing attribute has value "Standard" and whose Lifetime attribute has value "PERSISTENT" only if O.JAVAOBJECT's Context attribute has the same value as the active context.**
- **R.JAVA.3 ([JCRE22], §6.2.8.10): 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): S.PACKAGE may perform OP.INVK_INTERFACE upon an O.JAVAOBJECT whose Sharing attribute has the value "SIO", and whose Context attribute has the value "Package AID", only if the invoked interface method extends the Shareable interface and one of the following conditions applies:**

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

  b) The value of the attribute Selection Status of the package whose AID is "Package AID" is "Non-multiselectable", and either "Package AID" is the value of the currently selected applet or otherwise "Package AID" does not occur in the attribute Active Applets.
- **R.JAVA.5: S.PACKAGE may perform OP.CREATE only if the value of the Sharing parameter is "Standard"**.

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

- **1) The subject S.JCRE can freely perform OP.JAVA(") and OP.CREATE, with the exception given in FDP_ACF.1.4/FIREWALL, provided it is the Currently Active Context.**
- **2) The only means that the subject S.JCVM shall provide for an application to execute native code is the invocation of a Java Card API method (through OP.INVK_INTERFACE or OP.INVK_VIRTUAL)**.

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

- **1) Any subject with OP.JAVA upon an O.JAVAOBJECT whose LifeTime attribute has value "CLEAR_ON_DESELECT" if O.JAVAOBJECT's Context attribute is not the same as the Selected Applet Context.**
- **2) Any subject attempting to create an object by the means of OP.CREATE and a "CLEAR_ON_DESELECT" LifeTime parameter if the active context is not the same as the Selected Applet Context**.

Application Note:

FDP_ACF.1.4/FIREWALL: In the case of an array type, fields are components of the array ([JVM], §2.14, §2.7.7), as well as the length; the only methods of an array object are those inherited from the Object class.

The Sharing attribute defines four categories of objects:

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

When a new object is created, it is associated with the Currently Active Context. But the object is owned by the applet instance within the Currently Active Context when the object is instantiated ([JCRE22], §6.1.3). An object is owned by an applet instance, by the JCRE or by the package library where it has been defined (these latter objects can only be arrays that initialize static fields of packages).

([JCRE22], Glossary) Selected Applet Context. The Java Card RE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command with this applet's AID, the Java Card RE makes this applet the Selected Applet Context. The Java Card RE sends all APDU commands to the Selected Applet Context.

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

([JCRE22], §6.1.2.1) At any point in time, there is only one active context within the Java Card VM (this is called the Currently Active Context).

It should be noticed that the invocation of static methods (or access to a static field) is not considered by this policy, as there are no firewall rules. They have no effect on the active context as well and the "acting package" is not the one to which the static method belongs to in this case.

It should be noticed that the Java Card platform, version 2.2.x and version 3 Classic Edition, 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.

An applet instance will be considered an active applet instance if it is currently selected in at least one logical channel. 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).

FDP_IFC.1/JCVM Subset information flow control

FDP_IFC.1.1/JCVM The TSF shall enforce the **JCVM information flow control SFP** on **S.JCVM, S.LOCAL, S.MEMBER, I.DATA and OP.PUT(S1, S2, I)**.

Application Note:

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

FDP_IFF.1/JCVM Simple security attributes

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

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

.

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

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

FDP_IFF.1.3/JCVM The TSF shall enforce the **[assignment: additional information flow control SFP rules]**.

FDP_IFF.1.4/JCVM The TSF shall explicitly authorise an information flow based on the following rules: **[assignment: rules, based on security attributes, that explicitly authorise information flows]**.

FDP_IFF.1.5/JCVM The TSF shall explicitly deny an information flow based on the following rules: **[assignment: rules, based on security attributes, that explicitly deny information flows]**.

Application Note:

The storage of temporary Java Card RE-owned objects references is runtime-enforced ([JCRE22], §6.2.8.1-3).

It should be noticed that this policy essentially applies to the execution of bytecode. Native methods, the Java Card RE itself and possibly some API methods can be granted specific rights or limitations through the FDP_IFF.1.3/JCVM to FDP_IFF.1.5/JCVM elements. The way the Java Card virtual machine manages the transfer of values on the stack and local variables (returned values, uncaught exceptions) from and to internal registers is implementation-dependent. For instance, a returned reference, depending on the implementation of the stack frame, may transit through an internal register prior to being pushed on the stack of the invoker. The returned bytecode would cause more than one OP.PUT operation under this scheme.

FDP_RIP.1/OBJECTS 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/JCRE Management of security attributes

FMT_MSA.1.1/JCRE The TSF shall enforce the **FIREWALL access control SFP** to restrict the ability to **modify** the security attributes **Selected Applet Context** to **the Java Card RE**.

Application Note:

The modification of the Selected Applet Context should be performed in accordance with the rules given in [JCRE22], §4 and [JCVM22], §3.4.

FMT_MSA.1/JCVM Management of security attributes

FMT_MSA.1.1/JCVM The TSF shall enforce the **FIREWALL access control SFP and the JCVM information flow control SFP** to restrict the ability to **modify** the security attributes **Currently Active Context and Active Applets** to **the Java Card VM (S.JCVM)**.

Application Note:

The modification of the Currently Active Context should be performed in accordance with the rules given in [JCRE22], §4 and [JCVM22], §3.4.

FMT_MSA.2/FIREWALL_JCVM Secure security attributes

FMT_MSA.2.1/FIREWALL_JCVM The TSF shall ensure that only secure values are accepted for **all the security attributes of subjects and objects defined in the FIREWALL access control SFP and the JCVM information flow control SFP**.

Application Note:

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

- The Context attribute of an O.JAVAOBJECT must correspond to that of an installed applet or be "Java Card RE".
- An O.JAVAOBJECT whose Sharing attribute is a Java Card RE entry point or a global array necessarily has "Java Card RE" as the value for its Context security attribute.
- An O.JAVAOBJECT whose Sharing attribute value is a global array necessarily has "array of primitive 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.

FMT_MSA.3/FIREWALL Static attribute initialisation

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

FMT_MSA.3.2/FIREWALL [Editorially Refined] The TSF shall not allow **any role** to specify alternative initial values to override the default values when an object or information is created.

Application Note:

FMT_MSA.3.1/FIREWALL

- Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (FMT_MSA.1/JCRE). At the creation of an object (OP.CREATE), the newly created object, assuming that the FIREWALL access control SFP permits the operation, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([JCRE22], §6.1.3). There is one default value for the Selected Applet Context

that is the default applet identifier's Context, and one default value for the Currently Active Context that is "Java Card RE".

- The knowledge of which reference corresponds to a temporary entry point object or a global array and which does not is solely available to the Java Card RE (and the Java Card virtual machine).

FMT_MSA.3.2/FIREWALL

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

FMT_MSA.3/JCVM Static attribute initialisation

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

FMT_MSA.3.2/JCVM [Editorially Refined] The TSF shall not allow **any role** to specify alternative initial values to override the default values when an object or information is created.

FMT_SMF.1 Specification of Management Functions

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

- **modify the Currently Active Context, the Selected Applet Context and the Active Applets**.

FMT_SMR.1 Security roles

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

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

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

7.1.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

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

It should be noticed that the execution of the additional native code is not within the TSF. Nevertheless, access to API native methods from the Java Card System is controlled by TSF because there is no difference between native and interpreted methods in their interface or invocation mechanism.

FCS_CKM.1 Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **[assignment: cryptographic key generation algorithm]** and specified cryptographic key sizes **[assignment: cryptographic key sizes]** that meet the following: **[assignment: list of standards]**.

Application Note:

- The keys can be generated and diversified in accordance with [JCAPI22] specification in classes KeyBuilder and KeyPair (at least Session key generation).
- This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22], [JCAPI221], [JCAPI222] and [JCAPI3]).

FCS_CKM.2 Cryptographic key distribution

FCS_CKM.2.1 The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method **[assignment: cryptographic key distribution method]** that meets the following: **[assignment: list of standards]**.

Application Note:

- Command SetKEY that meets [JCAPI22] specification.
- This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI22], [JCAPI221], [JCAPI222] and [JCAPI3]).

FCS_CKM.3 Cryptographic key access

FCS_CKM.3.1 The TSF shall perform **[assignment: type of cryptographic key access]** in accordance with a specified cryptographic key access method **[assignment:**

cryptographic key access method] that meets the following: **[assignment: list of standards]**.

Application Note:

- The keys can be accessed as specified in [JCAPI22] Key class.
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([JCAPI22], [JCAPI221], [JCAPI222] and [JCAPI3]).

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 **[assignment: cryptographic key destruction method]** that meets the following: **[assignment: list of standards]**.

Application Note:

- The keys are reset as specified in [JCAPI22] Key class, with the method clearKey(). Any access to a cleared key for ciphering or signing shall throw an exception.
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([JCAPI22], [JCAPI221], [JCAPI222] and [JCAPI3]).

FCS_COP.1 Cryptographic operation

FCS_COP.1.1 The TSF shall perform **[assignment: list of cryptographic operations]** in accordance with a specified cryptographic algorithm **[assignment: cryptographic algorithm]** and cryptographic key sizes **[assignment: cryptographic key sizes]** that meet the following: **[assignment: list of standards]**.

Application Note:

- The TOE shall provide a subset of cryptographic operations defined in [JCAPI22] (see javacardx.crypto.Cipher and javacardx.security packages).
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([JCAPI22], [JCAPI221], [JCAPI222] and [JCAPI3]).

FDP_RIP.1/ABORT Subset residual information protection

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

Application Note:

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

FDP_RIP.1/APDU 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 objects: **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/bArray Subset residual information protection

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

Application Note:

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

FDP_RIP.1/KEYS Subset residual information protection

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

Application Note:

- The javacard.security & javacardx.crypto packages do provide secure interfaces to the cryptographic buffer in a transparent way. See javacard.security.KeyBuilder and Key interface of [JCAPI22].

FDP_RIP.1/TRANSIENT Subset residual information protection

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

Application Note:

- The events that provoke the de-allocation of any transient object are described in [JCRE22], §5.1.
- The clearing of CLEAR_ON_DESELECT objects is not necessarily performed when the owner of the objects is deselected. In the presence of multiselectable applet instances, CLEAR_ON_DESELECT memory segments may be attached to applets that are active in different logical channels. Multiselectable applet instances within a same package must share the transient memory segment if they are concurrently active ([JCRE22], §4.2.

FDP_ROL.1/FIREWALL 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 the **operations OP.JAVA and OP.CREATE** on the **object O.JAVAOBJECT**.

FDP_ROL.1.2/FIREWALL The TSF shall permit operations to be rolled back within the **scope of a select(), deselect(), process(), install() or uninstall() call, notwithstanding the restrictions given in [JCRE22], §7.7, within the bounds of the Commit Capacity ([JCRE22], §7.8), and those described in [JCAPI22]**.

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 [JCAPI22] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).

7.1.1.3 CARD SECURITY MANAGEMENT

FAU_ARP.1 Security alarms

FAU_ARP.1.1 The TSF shall take **one of the following actions:**

- **throw an exception,**
- **lock the card session,**
- **reinitialize the Java Card System and its data,**
- **[assignment: list of other actions]**

upon detection of a potential security violation.

Refinement:

The "potential security violation" stands for one of the following events:

- CAP file inconsistency,
- typing error in the operands of a bytecode,
- applet life cycle inconsistency,
- card tearing (unexpected removal of the Card out of the CAD) and power failure,
- abort of a transaction in an unexpected context, (see abortTransaction(), [JCAPI22] and ([JCRE22], §7.6.2)
- violation of the Firewall or JCVM SFPs,
- unavailability of resources,
- array overflow,
- [assignment: list of other runtime errors].

Application Note:

- The developer shall provide the exhaustive list of actual potential security violations the TOE reacts to. For instance, other runtime errors related to applet's failure like uncaught exceptions.
- 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.
- Depending on the context of use and the required security level, there are cases where the card manager and the TOE must work in cooperation to detect and appropriately react in case of potential security violation. This behavior must be described in this component. It shall detail the nature of the feedback information provided to the card manager (like the

identity of the offending application) and the conditions under which the feedback will occur (any occurrence of the java.lang.SecurityException exception).

- The "locking of the card session" may not appear in the policy of the card manager. Such measure should only be taken in case of severe violation detection; the same holds for the re-initialization of the Java Card System. Moreover, the locking should occur when "clean" re-initialization seems to be impossible.
- The locking may be implemented at the level of the Java Card System as a denial of service (through some systematic "fatal error" message or return value) that lasts up to the next "RESET" event, without affecting other components of the card (such as the card manager). Finally, because the installation of applets is a sensitive process, security alerts in this case should also be carefully considered herein.

FDP_SDI.2 Stored data integrity monitoring and action

FDP_SDI.2.1 The TSF shall monitor user data stored in containers controlled by the TSF for **[assignment: integrity errors]** on all objects, based on the following attributes: **[assignment: user data attributes]**.

FDP_SDI.2.2 Upon detection of a data integrity error, the TSF shall **[assignment: action to be taken]**.

Application Note:

- Although no such requirement is mandatory in the Java Card specification, at least an exception shall be raised upon integrity errors detection on cryptographic keys, PIN values and their associated security attributes. Even if all the objects cannot be monitored, cryptographic keys and PIN objects shall be considered with particular attention by ST authors as they play a key role in the overall security.
- It is also recommended to monitor integrity errors in the code of the native applications and Java Card applets.
- For integrity sensitive application, their data shall be monitored (D.APP_I_DATA): applications may need to protect information against unexpected modifications, and explicitly control whether a piece of information has been changed between two accesses. For example, maintaining the integrity of an electronic purse's balance is extremely important because this value represents real money. Its modification must be controlled, for illegal ones would denote an important failure of the payment system.
- A dedicated library could be implemented and made available to developers to achieve better security for specific objects, following the same pattern that already exists in cryptographic APIs, for instance.

FPR_UNO.1 Unobservability

FPR_UNO.1.1 The TSF shall ensure that **[assignment: list of users and/or subjects]** are unable to observe the operation **[assignment: list of operations]** on **[assignment: list of objects]** by **[assignment: list of protected users and/or subjects]**.

Application Note:

Although it is not required in [JCRE22] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.

FPT_FLS.1 Failure with preservation of secure state

FPT_FLS.1.1 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 Java Card RE Context is the Current context when the Java Card VM begins running after a card reset ([JCRE22], §6.2.3) or after a proximity card (PICC) activation sequence ([JCRE222]). Behavior of the TOE on power loss and reset is described in [JCRE22], §3.6 and §7.1. Behavior of the TOE on RF signal loss is described in [JCRE222], §3.6.1.

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, the bytecode and its data arguments** when shared between the TSF and another trusted IT product.

FPT_TDC.1.2 The TSF shall use

- **the rules defined in [JCVM22] specification,**
- **the API tokens defined in the export files of reference implementation,**
- **[assignment: list of interpretation rules to be applied by the TSF]**

when interpreting the TSF data from 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, including memory management, I/O functions and cryptographic functions.

7.1.1.4 AID MANAGEMENT

FIA_ATD.1/AID User attribute definition

FIA_ATD.1.1/AID The TSF shall maintain the following list of security attributes belonging to individual users:

- **Package AID,**
- **Applet's version number,**
- **Registered applet AID,**
- **Applet Selection Status ([JCVM22], §6.5)**.

Refinement:

"Individual users" stand for applets.

FIA_UID.2/AID User identification before any action

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

Application Note:

- By users here it must be understood the ones associated to the packages (or applets) that act as subjects of policies. In the Java Card System, every action is always performed by an identified user interpreted here as the currently selected applet or the package that is the subject's owner. Means of identification are provided during the loading procedure of the package and the registration of applet instances.
- The role Java Card RE defined in FMT_SMR.1 is attached to an IT security function rather than to a "user" of the CC terminology. The Java Card RE does not "identify" itself to the TOE, but it is part of it.

FIA_USB.1/AID User-subject binding

FIA_USB.1.1/AID The TSF shall associate the following user security attributes with subjects acting on the behalf of that user: **Package AID**.

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

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

Application Note:

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

FMT_MTD.1/JCRE Management of TSF data

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

Application Note:

- The Java Card RE manages other TSF data such as the applet life cycle or CAP files, but this management is implementation specific. Objects in the Java programming language may also try to query AIDs of installed applets through the lookupAID(...) API method.

FMT_MTD.3/JCRE Secure TSF data

FMT_MTD.3.1/JCRE The TSF shall ensure that only secure values are accepted for **the registered applets' AIDs**.

7.1.2 RMIG Security Functional Requirements

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

global arrays. Therefore, the storage of references to those arrays must be restricted as well. The JCRMI policy embodies both an access control and an information flow control policy.

This group of SFRs applies only if the TOE provides JCRMI functionality. Otherwise, it shall be ignored.

FDP_ACC.2/JCRMI 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.

Refinement:

The operations involved in this policy are:

- OP.GET_ROR,
- OP.INVOKE.

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

FDP_ACF.1/JCRMI 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:

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

.

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

- **R.JAVA.18: S.CAD may perform OP.GET_ROR upon O.APPLET only if O.APPLET is the currently selected applet, and there exists an O.RMI_SERVICE with a registered initial reference to an O.REMOTE_OBJ that is owned by O.APPLET.**
- **R.JAVA.19: S.JCRE may perform OP.INVOKE upon O.RMI_SERVICE, O.ROR and O.REMOTE_MTHD only if O.ROR is valid (as defined in [JCRE22], §8.5) and it belongs to the Returned References of O.RMI_SERVICE, and if the Identifier**

of O.REMOTE_MTHD matches one of the remote methods in the Class of the O.REMOTE_OBJ to which O.ROR makes reference.

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

FDP_ACF.1.4/JCRMI [Editorially Refined] The TSF shall explicitly deny access of **any subject but S.JCRE to O.REMOTE_OBJ and O.REMOTE_MTHD for the purpose of performing a remote method invocation**.

Application Note:

FDP_ACF.1.2/JCRMI:

- The validity of a remote object reference is specified as a lifetime characterization. The security attributes involved in the rules for determining valid remote object references are the Returned References of the O.RMI_SERVICE and the Active Applets (see FMT_REV.1.1/JCRMI and FMT_REV.1.2/JCRMI). The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([JCRE22], §8.5.2 and [JCAPI22]).
- Note that 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 S.CAD during the applet selection session. This attribute is implementation dependent.

FDP_IFC.1/JCRMI Subset information flow control

FDP_IFC.1.1/JCRMI The TSF shall enforce the **JCRMI information flow control SFP** on **S.JCRE, S.CAD, I.RORD and OP.RET_RORD(S.JCRE,S.CAD,I.RORD)**.

Application Note:

FDP_IFC.1.1/JCRMI:

- Array parameters of remote method invocations must be allocated on the card as global arrays objects. References to global arrays cannot be stored in class variables, instance variables or array components. The control of the flow of that kind of information has already been specified in FDP_IFC.1.1/JCVM.
- A remote object reference descriptor is sent from the card to the CAD either as the result of a successful applet selection command ([JCRE22], §8.4.1), and in this case it describes, if any, the initial remote object reference of the selected applet; or as the result of a remote method invocation ([JCRE22],§8.3.5.1).

FDP_IFF.1/JCRMI 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:

Subjects/Information	Security attributes
I.RORD	**ExportedInfo**

.

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

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

FDP_IFF.1.3/JCRMI The TSF shall enforce the **[assignment: additional information flow control SFP rules]**.

FDP_IFF.1.4/JCRMI The TSF shall explicitly authorise an information flow based on the following rules: **[assignment: rules, based on security attributes, that explicitly authorise information flows]**.

FDP_IFF.1.5/JCRMI The TSF shall explicitly deny an information flow based on the following rules: **[assignment: rules, based on security attributes, that explicitly deny information flows]**.

Application Note:

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

FMT_MSA.1/EXPORT Management of security attributes

FMT_MSA.1.1/EXPORT The TSF shall enforce the **JCRMI access control SFP** to restrict the ability to **modify** the security attributes**: ExportedInfo of O.REMOTE_OBJ** to **its owner applet**.

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 can be performed by the Java Card RE.

FMT_MSA.1/REM_REFS Management of security attributes

FMT_MSA.1.1/REM_REFS The TSF shall enforce the **JCRMI access control SFP** to restrict the ability to **modify** the security attributes **Returned References of O.RMI_SERVICE** to **its owner applet**.

FMT_MSA.3/JCRMI Static attribute initialisation

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.

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

Application Note:

FMT_MSA.3.1/JCRMI:

- Remote objects' security attributes are created and initialized at the creation of the object, and except for the ExportedInfo attribute, the values of the attributes are not longer modifiable. The default value of the Exported attribute is true. There is one default value for the Selected Applet Context that is the default applet identifier's context, and one default value for the active context, that is "Java Card RE".

FMT_MSA.3.2/JCRMI:

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

FMT_REV.1/JCRMI Revocation

FMT_REV.1.1/JCRMI [Editorially Refined] The TSF shall restrict the ability to revoke **the Returned References of O.RMI_SERVICE** to **the Java Card RE**.

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

Application Note:

The rules are described in [JCRE22], §8.5

FMT_SMF.1/JCRMI Specification of Management Functions

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

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

FMT_SMR.1/JCRMI Security roles

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

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

Application Note:

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

7.1.3 ODELG SECURITY FUNCTIONAL REQUIREMENTS

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

FDP_RIP.1/ODEL 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 **deallocation of the resource from** the following objects: **the objects owned by the context of an applet instance which triggered the execution of the method** `javacard.framework.JCSystem.requestObjectDeletion()`.

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].
- 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/ODEL Failure with preservation of secure state

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

Application Note:

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

7.2 SECURITY ASSURANCE REQUIREMENTS

The Evaluation Assurance Level is EAL4 augmented with ALC_DVS.2 and AVA_VAN.5.

7.3 SECURITY REQUIREMENTS RATIONALE

7.3.1 OBJECTIVES

7.3.1.1 SECURITY OBJECTIVES FOR THE TOE

7.3.1.1.1 IDENTIFICATION

O.SID Subjects' identity is AID-based (applets, packages), and is met by the following SFRs: FIA_ATD.1/AID, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_MSA.1/REM_REFS, FMT_MSA.1/EXPORT, FMT_MSA.3/FIREWALL, FMT_MSA.3/JCVM, FMT_MTD.1/JCRE and FMT_MTD.3/JCRE.

Additionally, if the TOE provides JCRMI functionality, subjects' identity is also met by the following SFRs: FMT_MSA.3/JCRMI, FMT_SMF.1/JCRMI, FMT_SMR.1/JCRMI.

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/AID, FIA_USB.1/AID).

7.3.1.1.2 EXECUTION

O.FIREWALL This objective is met by the FIREWALL access control policy (FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL) and the JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM). The functional requirements of the class FMT (FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_SMR.1, FMT_SMF.1, FMT_MSA.2/FIREWALL_JCVM, FMT_MSA.3/FIREWALL, FMT_MSA.3/JCVM, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM) also indirectly contribute to meet this objective.

Additionally, if the TOE provides JCRMI functionality, this objective is met by the JCRMI access control policy (FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI). The functional requirements of the class FMT (FMT_SMR.1/JCRMI, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_SMF.1/JCRMI, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI) also indirectly contribute to meet this objective.

O.GLOBAL_ARRAYS_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 global byte array input parameter (bArray) to an applet's install method. The clearing requirement of these arrays is met by (FDP_RIP.1/APDU and FDP_RIP.1/bArray respectively). The JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM) 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.

If the TOE provides JCRMI functionality, 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/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS and FDP_RIP.1/TRANSIENT).

O.GLOBAL_ARRAYS_INTEG This objective is met by the JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM), which prevents an application from keeping a pointer to the APDU buffer of the card or to the global byte array of the applet's install method. Such a pointer could be used to access and modify it when the buffer is being used by another application.

O.NATIVE This security objective is covered by FDP_ACF.1/FIREWALL: the only means to execute native code is the invocation of a Java Card API method.

O.OPERATE The TOE is protected in various ways against applets' actions (FPT_TDC.1), the FIREWALL access control policy FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL, and is able to detect and block various failures or security violations during usual working (FPT_FLS.1, FPT_FLS.1/ODEL, FAU_ARP.1). Its security-critical parts and procedures are also protected: applets' installation may be cleanly aborted (FDP_ROL.1/FIREWALL), communication with external users and their internal subjects is well-controlled (FIA_ATD.1/AID, FIA_USB.1/AID) 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.

Application note: Startup of the TOE (TSF-testing) can be covered by FPT_TST.1. This SFR component is not mandatory in [JCRE22], but appears in most of security requirements

documents for masked applications. Testing could also occur randomly. Self-tests may become mandatory in order to comply with FIPS certification [FIPS 140-2].

O.REALLOCATION This security objective is satisfied by the following SFRs: FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, which imposes that the contents of the re-allocated block shall always be cleared before delivering the block.

O.RESOURCES The TSFs detects stack/memory overflows during execution of applications (FAU_ARP.1, FPT_FLS.1, FPT_FLS.1/ODEL). Failed installations are not to create memory leaks (FDP_ROL.1/FIREWALL) as well. Memory management is controlled by the TSF (FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_SMR.1, FMT_SMF.1).

Additionally, if the TOE provides JCRMI functionality, memory management is controlled by the TSF FMT_SMR.1/JCRMI, and FMT_SMF.1/JCRMI.

7.3.1.1.3 SERVICES

O.ALARM This security objective is met by FPT_FLS.1, FPT_FLS.1/ODEL which guarantee that a secure state is preserved by the TSF when failures occur, and FAU_ARP.1 which defines TSF reaction upon detection of a potential security violation.

O.CIPHER This security objective is directly covered by FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4 and FCS_COP.1. The SFR FPR_UNO.1 contributes in covering this security objective and controls the observation of the cryptographic operations which may be used to disclose the keys.

O.KEY-MNGT This relies on the same security functional requirements as O.CIPHER, plus FDP_RIP.1 and FDP_SDI.2 as well. Precisely it is met by the following components: FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1, FPR_UNO.1, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT and FDP_SDI.2.

O.PIN-MNGT This security objective is ensured by FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT, FPR_UNO.1, FDP_ROL.1/FIREWALL and FDP_SDI.2 security functional requirements. The TSFs behind these are implemented by API classes. The firewall security functions FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL shall protect the access to private and internal data of the objects.

O.REMOTE If the TOE provides JCRMI functionality, 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 security functional requirements of the class FMT (FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI and FMT_SMR.1/JCRMI) included in the group RMIG also contribute to meet this objective.

O.TRANSACTION Directly met by FDP_ROL.1/FIREWALL, FDP_RIP.1/ABORT (more precisely, by the element FDP_RIP.1.1/ABORT), FDP_RIP.1/ODEL, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT and FDP_RIP.1/OBJECTS.

7.3.1.1.4 OBJECT DELETION

O.OBJ-DELETION This security objective specifies that deletion of objects is secure. The security objective is met by the security functional requirements FDP_RIP.1/ODEL and FPT_FLS.1/ODEL.

7.3.2 RATIONALE TABLES OF SECURITY OBJECTIVES AND SFRS

Security Objectives	Security Functional Requirements	Rationale
O.SID	FIA_ATD.1/AID, FIA_UID.2/AID, FMT_MSA.1/JCRE, FMT_MSA.3/JCRMI,	Section 7.3.3.1

	FMT_MSA.1/REM_REFS, FMT_MSA.1/EXPORT, FMT_MSA.3/FIREWALL, FMT_SMF.1/JCRMI, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FIA_USB.1/AID, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM, FMT_SMR.1/JCRMI	
O.FIREWALL	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FMT_MSA.3/FIREWALL, FMT_SMR.1, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI, FMT_MSA.1/JCRE, FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FMT_SMF.1/JCRMI, FMT_SMF.1, FMT_MSA.2/FIREWALL_JCVM, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM	Section 7.3.3.1
O.GLOBAL_ARRAYS_CONFID	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FDP_RIP.1/bArray, FDP_RIP.1/APDU, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT	Section 7.3.3.1
O.GLOBAL_ARRAYS_INTEG	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM	Section 7.3.3.1
O.NATIVE	FDP_ACF.1/FIREWALL	Section 7.3.3.1
O.OPERATE	FAU_ARP.1, FDP_ROL.1/FIREWALL, FIA_ATD.1/AID, FPT_FLS.1, FPT_FLS.1/ODEL, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FPT_TDC.1, FIA_USB.1/AID	Section 7.3.3.1
O.REALLOCATION	FDP_RIP.1/ABORT, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS	Section 7.3.3.1
O.RESOURCES	FAU_ARP.1, FDP_ROL.1/FIREWALL, FMT_SMR.1, FMT_SMR.1/JCRMI, FPT_FLS.1/ODEL, FPT_FLS.1, FMT_SMF.1/JCRMI, FMT_SMF.1, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE	Section 7.3.3.1
O.ALARM	FPT_FLS.1, FPT_FLS.1/ODEL, FAU_ARP.1	Section 7.3.3.1
O.CIPHER	FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1, FPR_UNO.1	Section 7.3.3.1

O.KEY-MNGT	FCS_CKM.1, FCS_CKM.2, FCS_CKM.3, FCS_CKM.4, FCS_COP.1, FPR_UNO.1, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_SDI.2, FDP_RIP.1/TRANSIENT	Section 7.3.3.1
O.PIN-MNGT	FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FPR_UNO.1, FDP_RIP.1/TRANSIENT, FDP_ROL.1/FIREWALL, FDP_SDI.2, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL	Section 7.3.3.1
O.REMOTE	FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI	Section 7.3.3.1
O.TRANSACTION	FDP_ROL.1/FIREWALL, FDP_RIP.1/ABORT, FDP_RIP.1/ODEL, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT, FDP_RIP.1/OBJECTS	Section 7.3.3.1
O.OBJ-DELETION	FDP_RIP.1/ODEL, FPT_FLS.1/ODEL	Section 7.3.3.1

Table 7 Security Objectives and SFRs - Coverage

Security Functional Requirements	Security Objectives
FDP_ACC.2/FIREWALL	O.FIREWALL, O.OPERATE, O.PIN-MNGT
FDP_ACF.1/FIREWALL	O.FIREWALL, O.NATIVE, O.OPERATE, O.PIN-MNGT
FDP_IFC.1/JCVM	O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.GLOBAL_ARRAYS_INTEG
FDP_IFF.1/JCVM	O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.GLOBAL_ARRAYS_INTEG
FDP_RIP.1/OBJECTS	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FMT_MSA.1/JCRE	O.SID, O.FIREWALL
FMT_MSA.1/JCVM	O.SID, O.FIREWALL
FMT_MSA.2/FIREWALL_JCVM	O.FIREWALL
FMT_MSA.3/FIREWALL	O.SID, O.FIREWALL
FMT_MSA.3/JCVM	O.SID, O.FIREWALL
FMT_SMF.1	O.FIREWALL, O.RESOURCES
FMT_SMR.1	O.FIREWALL, O.RESOURCES
FCS_CKM.1	O.CIPHER, O.KEY-MNGT
FCS_CKM.2	O.CIPHER, O.KEY-MNGT
FCS_CKM.3	O.CIPHER, O.KEY-MNGT
FCS_CKM.4	O.CIPHER, O.KEY-MNGT
FCS_COP.1	O.CIPHER, O.KEY-MNGT
FDP_RIP.1/ABORT	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/APDU	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/bArray	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/KEYS	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/TRANSIENT	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-

	MNGT, O.TRANSACTION
FDP_ROL.1/FIREWALL	O.OPERATE, O.RESOURCES, O.PIN-MNGT, O.TRANSACTION
FAU_ARP.1	O.OPERATE, O.RESOURCES, O.ALARM
FDP_SDI.2	O.KEY-MNGT, O.PIN-MNGT
FPR_UNO.1	O.CIPHER, O.KEY-MNGT, O.PIN-MNGT
FPT_FLS.1	O.OPERATE, O.RESOURCES, O.ALARM
FPT_TDC.1	O.OPERATE
FIA_ATD.1/AID	O.SID, O.OPERATE
FIA_UID.2/AID	O.SID
FIA_USB.1/AID	O.SID, O.OPERATE
FMT_MTD.1/JCRE	O.SID, O.FIREWALL, O.RESOURCES
FMT_MTD.3/JCRE	O.SID, O.FIREWALL, O.RESOURCES
FDP_ACC.2/JCRMI	O.FIREWALL, O.REMOTE
FDP_ACF.1/JCRMI	O.FIREWALL, O.REMOTE
FDP_IFC.1/JCRMI	O.REMOTE
FDP_IFF.1/JCRMI	O.REMOTE
FMT_MSA.1/EXPORT	O.SID, O.FIREWALL, O.REMOTE
FMT_MSA.1/REM_REFS	O.SID, O.FIREWALL, O.REMOTE
FMT_MSA.3/JCRMI	O.SID, O.FIREWALL, O.REMOTE
FMT_REV.1/JCRMI	O.FIREWALL, O.REMOTE
FMT_SMF.1/JCRMI	O.SID, O.FIREWALL, O.RESOURCES
FMT_SMR.1/JCRMI	O.SID, O.FIREWALL, O.RESOURCES, O.REMOTE
FDP_RIP.1/ODEL	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION, O.OBJ-DELETION
FPT_FLS.1/ODEL	O.OPERATE, O.RESOURCES, O.ALARM, O.OBJ-DELETION

Table 8 SFRs and Security Objectives

7.3.3 DEPENDENCIES

7.3.3.1 SFRs DEPENDENCIES

Requirements	CC Dependencies	Satisfied Dependencies
FDP_ACC.2/JCRMI	(FDP_ACF.1)	FDP_ACF.1/JCRMI
FDP_ACF.1/JCRMI	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.2/JCRMI, FMT_MSA.3/JCRMI
FDP_IFC.1/JCRMI	(FDP_IFF.1)	FDP_IFF.1/JCRMI
FDP_IFF.1/JCRMI	(FDP_IFC.1) and (FMT_MSA.3)	FDP_IFC.1/JCRMI, FMT_MSA.3/JCRMI
FMT_MSA.1/EXPORT	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/JCRMI, FMT_SMF.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.1/REM_REFS	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/JCRMI, FMT_SMF.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.3/JCRMI	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_SMR.1/JCRMI
FMT_REV.1/JCRMI	(FMT_SMR.1)	FMT_SMR.1/JCRMI
FMT_SMF.1/JCRMI	No Dependencies	
FMT_SMR.1/JCRMI	(FIA_UID.1)	FIA_UID.2/AID
FDP_RIP.1/ODEL	No Dependencies	
FPT_FLS.1/ODEL	No Dependencies	
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	FDP_ACF.1/FIREWALL
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL
FDP_IFC.1/JCVM	(FDP_IFF.1)	FDP_IFF.1/JCVM
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	FDP_IFC.1/JCVM, FMT_MSA.3/JCVM
FDP_RIP.1/OBJECTS	No Dependencies	
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/FIREWALL, FMT_SMR.1
FMT_MSA.1/JCVM	(FDP_ACC.1 or FDP_IFC.1) and	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMF.1,

	(FMT_SMF.1) and (FMT_SMR.1)	FMT_SMR.1
FMT_MSA.2/FIREWALL_JCVM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_SMR.1
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_SMR.1
FMT_MSA.3/JCVM	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/JCVM, FMT_SMR.1
FMT_SMF.1	No Dependencies	
FMT_SMR.1	(FIA_UID.1)	FIA_UID.2/AID
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4)	FCS_CKM.2, FCS_CKM.4
FCS_CKM.2	(FCS_CKM.1 or FDP_ITC.1 or FDP_ITC.2) and (FCS_CKM.4)	FCS_CKM.1, FCS_CKM.4
FCS_CKM.3	(FCS_CKM.1 or FDP_ITC.1 or FDP_ITC.2) and (FCS_CKM.4)	FCS_CKM.1, FCS_CKM.4
FCS_CKM.4	(FCS_CKM.1 or FDP_ITC.1 or FDP_ITC.2)	FCS_CKM.1
FCS_COP.1	(FCS_CKM.1 or FDP_ITC.1 or FDP_ITC.2) and (FCS_CKM.4)	FCS_CKM.1, FCS_CKM.4
FDP_RIP.1/ABORT	No Dependencies	
FDP_RIP.1/APDU	No Dependencies	
FDP_RIP.1/bArray	No Dependencies	
FDP_RIP.1/KEYS	No Dependencies	
FDP_RIP.1/TRANSIENT	No Dependencies	
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM
FAU_ARP.1	(FAU_SAA.1)	
FDP_SDI.2	No Dependencies	
FPR_UNO.1	No Dependencies	
FPT_FLS.1	No Dependencies	

FPT_TDC.1	No Dependencies	
FIA_ATD.1/AID	No Dependencies	
FIA_UID.2/AID	No Dependencies	
FIA_USB.1/AID	(FIA_ATD.1)	FIA_ATD.1/AID
FMT_MTD.1/JCRE	(FMT_SMF.1) and (FMT_SMR.1)	FMT_SMF.1, FMT_SMR.1
FMT_MTD.3/JCRE	(FMT_MTD.1)	FMT_MTD.1/JCRE

Table 9 SFRs Dependencies

7.3.3.1.1 RATIONALE FOR THE EXCLUSION OF DEPENDENCIES

The dependency FMT_SMF.1 of FMT_MSA.1/JCRE is discarded. The dependency between FMT_MSA.1/JCRE and FMT_SMF.1 is not satisfied because no management functions are required for the Java Card RE.

The dependency FAU_SAA.1 of FAU_ARP.1 is discarded. The dependency of FAU_ARP.1 on FAU_SAA.1 assumes that a "potential security violation" generates an audit event. On the contrary, the events listed in FAU_ARP.1 are self-contained (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, there is no mandatory audit recording in this PP.

7.3.3.2 SARS DEPENDENCIES

Requirements	CC Dependencies	Satisfied Dependencies
ADV_ARC.1	(ADV_FSP.1) and (ADV_TDS.1)	ADV_FSP.4, ADV_TDS.3
ADV_FSP.4	(ADV_TDS.1)	ADV_TDS.3
ADV_IMP.1	(ADV_TDS.3) and (ALC_TAT.1)	ADV_TDS.3, ALC_TAT.1
ADV_TDS.3	(ADV_FSP.4)	ADV_FSP.4
AGD_OPE.1	(ADV_FSP.1)	ADV_FSP.4
AGD_PRE.1	No Dependencies	
ALC_CMC.4	(ALC_CMS.1) and (ALC_DVS.1) and (ALC_LCD.1)	ALC_CMS.4, ALC_DVS.2, ALC_LCD.1
ALC_CMS.4	No Dependencies	
ALC_DEL.1	No Dependencies	
ALC_DVS.2	No Dependencies	
ALC_LCD.1	No Dependencies	
ALC_TAT.1	(ADV_IMP.1)	ADV_IMP.1
ASE_CCL.1	(ASE_ECD.1) and (ASE_INT.1) and (ASE_REQ.1)	ASE_ECD.1, ASE_INT.1, ASE_REQ.2
ASE_ECD.1	No Dependencies	
ASE_INT.1	No Dependencies	
ASE_OBJ.2	(ASE_SPD.1)	ASE_SPD.1
ASE_REQ.2	(ASE_ECD.1) and (ASE_OBJ.2)	ASE_ECD.1, ASE_OBJ.2
ASE_SPD.1	No Dependencies	
ASE_TSS.1	(ADV_FSP.1) and (ASE_INT.1) and (ASE_REQ.1)	ADV_FSP.4, ASE_INT.1, ASE_REQ.2
ATE_COV.2	(ADV_FSP.2) and (ATE_FUN.1)	ADV_FSP.4, ATE_FUN.1

ATE_DPT.1	(ADV_ARC.1) and (ADV_TDS.2) and (ATE_FUN.1)	ADV_ARC.1, ADV_TDS.3, ATE_FUN.1
ATE_FUN.1	(ATE_COV.1)	ATE_COV.2
ATE_IND.2	(ADV_FSP.2) and (AGD_OPE.1) and (AGD_PRE.1) and (ATE_COV.1) and (ATE_FUN.1)	ADV_FSP.4, AGD_OPE.1, AGD_PRE.1, ATE_COV.2, ATE_FUN.1
AVA_VAN.5	(ADV_ARC.1) and (ADV_FSP.4) and (ADV_IMP.1) and (ADV_TDS.3) and (AGD_OPE.1) and (AGD_PRE.1) and (ATE_DPT.1)	ADV_ARC.1, ADV_FSP.4, ADV_IMP.1, ADV_TDS.3, AGD_OPE.1, AGD_PRE.1, ATE_DPT.1

Table 10 SARs Dependencies

7.3.4 Rationale for the Security Assurance Requirements

EAL4 is required for this type of TOE and product since it is intended to defend against sophisticated attacks. This evaluation assurance level allows a developer to gain maximum assurance from positive security engineering based on good practices. EAL4 represents the highest practical level of assurance expected for a commercial grade product. In order to provide a meaningful level of assurance that the TOE and its embedding product provide an adequate level of defense against such attacks: the evaluators should have access to the low level design and source code. The lowest for which such access is required is EAL4.

7.3.5 ALC_DVS.2 Sufficiency of security measures

Development security is concerned with physical, procedural, personnel and other technical measures that may be used in the development environment to protect the TOE and the embedding product. The standard ALC_DVS.1 requirement mandated by EAL4 is not enough. Due to the nature of the TOE and embedding product, it is necessary to justify the sufficiency of these procedures to protect their confidentiality and integrity. ALC_DVS.2 has no dependencies.

7.3.6 AVA_VAN.5 Advanced methodical vulnerability analysis

The TOE is intended to operate in hostile environments. AVA_VAN.5 "Advanced methodical vulnerability analysis" is considered as the expected level for Java Card technology-based products hosting sensitive applications, in particular in payment and identity areas. AVA_VAN.5 has dependencies on ADV_ARC.1, ADV_FSP.1, ADV_TDS.3, ADV_IMP.1, AGD_PRE.1 and AGD_OPE.1. All of them are satisfied by EAL4.

8 NOTICE

This document has been generated with TL SET version 3.0.1-Full (for CC3). For more information about the security editor tool of Trusted Labs visit our website at www.trusted-labs.com.

APPENDIX 1: OVERVIEW AND COMPARISON OF CLOSED AND OPEN CONFIGURATIONS

This section provides an all-embracing presentation that compares the security problem definition, the security objectives and the security functional requirements of the JCS Open and Closed Configurations Protection Profiles. It is based on this document for the Closed Configuration and on the document [PP-JCSO-3.0] for the Open Configuration.

Assets are common to all configurations.

The configurations' assumptions are displayed in Table A3-1.

Assumption	*Closed*	*Open*
A.NO-INSTALL	**X**	
A.NO-DELETION	**X**	
A.APPLET		**X**
A.DELETION		**X**
A.VERIFICATION	**X**	**X**

Table A3-1: Assumptions by Configuration

The threats to the assets against which specific protection is required within the configurations or their environments are displayed in Table A3-2.

Threat	*Closed*	*Open*
T.PHYSICAL	**X**	**X**
T.CONFID-JCS-CODE	**X**	**X**
T.CONFID-APPLI-DATA	**X**	**X**
T.CONFID-JCS-DATA	**X**	**X**
T.INTEG-APPLI-CODE	**X**	**X**
T.INTEG-JCS-CODE	**X**	**X**
T.INTEG-APPLI-DATA	**X**	**X**
T.INTEG-JCS-DATA	**X**	**X**
T.SID.1	**X**	**X**
T.SID.2	**X**	**X**
T.EXE-CODE.1	**X**	**X**
T.EXE-CODE.2	**X**	**X**
T.NATIVE	**X**	**X**
T.RESOURCES	**X**	**X**
T.INTEG-APPLI-CODE.LOAD	**X (restricted to pre-issuance phase)**	**X**
T.INTEG-APPLI-DATA.LOAD	**X (restricted to pre-issuance phase)**	**X**
T.INSTALL		**X**
T.EXE-CODE-REMOTE	**X**	**X**
T.DELETION		**X**
T.OBJ-DELETION	**X**	**X**

Table A3-2: Threats by Configuration

Table A3-3 lists the security objectives addressed by each of these TOEs.

TOE security objective	*Closed*	*Open*
O.SID	**X**	**X**
O.OPERATE	**X**	**X**
O.RESOURCES	**X**	**X**
O.FIREWALL	**X**	**X**
O.NATIVE	**X**	**X**
O.REALLOCATION	**X**	**X**
O.GLOBAL_ARRAYS_CONFID	**X**	**X**
O.GLOBAL_ARRAYS_INTEG	**X**	**X**
O.ALARM	**X**	**X**
O.TRANSACTION	**X**	**X**
O.CIPHER	**X**	**X**
O.PIN-MNGT	**X**	**X**
O.KEY-MNGT	**X**	**X**
O.INSTALL		**X**
O.LOAD		**X**
O.DELETION		**X**
O.OBJ-DELETION	**X**	**X**
O.REMOTE	**X**	**X**
O.BIO-MNGT	**X (Closed 2.2.2 optional feature)**	**X (Open 2.2.2 optional feature)**
O.EXT-MEM	**X (Closed 2.2.2 optional feature)**	**X (Open 2.2.2 optional feature)**

Table A3-3: TOE Security Objectives by Configuration

Table A3-4 displays the security objectives to be achieved by the environment associated to each TOE configuration.

Environment security objective	*Closed*	*Open*
OE.SCP.RECOVERY	**X**	**X**
OE.SCP.SUPPORT	**X**	**X**
OE.SCP.IC	**X**	**X**
OE.NO-DELETION	**X**	
OE.NO-INSTALL	**X**	
OE.VERIFICATION	**X**	**X**
OE.APPLET		**X**
OE.CARD-MANAGEMENT	**X**	**X**
OE.CODE-EVIDENCE	**X**	**X**

Table A3-4: Security objectives for the environment by Configuration

Table A3-5 states the relationship between the SFRs, the groups of to which they belong, and the JCS configurations defined in this document.

SFR	*Group*	*Closed*	*Open*
FAU_ARP.1	*CoreG_LC*	X	X
FCS_CKM.1	*CoreG_LC*	X	X
FCS_CKM.2	*CoreG_LC*	X	X
FCS_CKM.3	*CoreG_LC*	X	X
FCS_CKM.4	*CoreG_LC*	X	X
FCS_COP.1	*CoreG_LC*	X	X
FDP_ACC.2/FIREWALL	*CoreG_LC*	X	X
FDP_ACF.1/FIREWALL	*CoreG_LC*	X	X
FDP_IFC.1/JCVM	*CoreG_LC*	X	X
FDP_IFF.1/JCVM	*CoreG_LC*	X	X
FDP_RIP.1/ABORT	*CoreG_LC*	X	X
FDP_RIP.1/APDU	*CoreG_LC*	X	X
FDP_RIP.1/bArray	*CoreG_LC*	X	X
FDP_RIP.1/KEYS	*CoreG_LC*	X	X
FDP_RIP.1/TRANSIENT	*CoreG_LC*	X	X
FDP_RIP.1/OBJECTS	*CoreG_LC*	X	X
FDP_ROL.1/FIREWALL	*CoreG_LC*	X	X
FDP_SDI.2	*CoreG_LC*	X	X
FIA_ATD.1/AID	*CoreG_LC*	X	X
FIA_UID.2/AID	*CoreG_LC*	X	X
FIA_USB.1/AID	*CoreG_LC*	X	X
FMT_MSA.1/JCRE	*CoreG_LC*	X	X
FMT_MSA.1/JCVM	*CoreG_LC*	X	X
FMT_MSA.2/FIREWALL_JCVM	*CoreG_LC*	X	X
FMT_MSA.3/JCVM	*CoreG_LC*	X	X
FMT_MSA.3/FIREWALL	*CoreG_LC*	X	X
FMT_MTD.1/JCRE	*CoreG_LC*	X	X
FMT_MTD.3/JCRE	*CoreG_LC*	X	X
FMT_SMR.1	*CoreG_LC*	X	X
FMT_SMF.1	*CoreG_LC*	X	X
FPR_UNO.1	*CoreG_LC*	X	X
FPT_FLS.1	*CoreG_LC*	X	X
FPT_TDC.1	*CoreG_LC*	X	X
FDP_ITC.2/Installer	*InstG*		X
FMT_SMR.1/Installer	*InstG*		X
FPT_FLS.1/Installer	*InstG*		X

FPT_RCV.3/Installer	*InstG*		**X**
FDP_ACC.2/ADEL	*ADELG*		**X**
FDP_ACF.1/ADEL	*ADELG*		**X**
FMT_MSA.1/ADEL	*ADELG*		**X**
FMT_MSA.3/ADEL	*ADELG*		**X**
FMT_SMR.1/ADEL	*ADELG*		**X**
FMT_SMF.1/ADEL	*ADELG*		**X**
FDP_RIP.1/ADEL	*ADELG*		**X**
FPT_FLS.1/ADEL	*ADELG*		**X**
FDP_ACC.2/JCRMI	*RMIG*	**X**	**X**
FDP_ACF.1/JCRMI	*RMIG*	**X**	**X**
FDP_IFC.1/JCRMI	*RMIG*	**X**	**X**
FDP_IFF.1/JCRMI	*RMIG*	**X**	**X**
FMT_MSA.1/EXPORT	*RMIG*	**X**	**X**
FMT_MSA.1/REM_REFS	*RMIG*	**X**	**X**
FMT_MSA.3/JCRMI	*RMIG*	**X**	**X**
FMT_REV.1/JCRMI	*RMIG*	**X**	**X**
FMT_SMR.1/JCRMI	*RMIG*	**X**	**X**
FMT_SMF.1/JCRMI	*RMIG*	**X**	**X**
FDP_RIP.1/ODEL	*ODELG*	**X**	**X**
FPT_FLS.1/ODEL	*ODELG*	**X**	**X**
FCO_NRO.2/CM	*CarG*		**X**
FDP_IFC.2/CM	*CarG*		**X**
FDP_IFF.1/CM	*CarG*		**X**
FDP_UIT.1/CM	*CarG*		**X**
FMT_MSA.1/CM	*CarG*		**X**
FMT_MSA.3/CM	*CarG*		**X**
FMT_SMR.1/CM	*CarG*		**X**
FIA_UID.1/CM	*CarG*		**X**
FTP_ITC.1/CM	*CarG*		**X**
FMT_SMF.1/CM	*CarG*		**X**
FDP_ACC.1/EXT_MEM	*EMG*		**X**
FDP_ACF.1/EXT_MEM	*EMG*		**X**
FMT_MSA.1/EXT_MEM	*EMG*		**X**
FMT_MSA.3/EXT_MEM	*EMG*		**X**
FMT_SMF.1/EXT_MEM	*EMG*		**X**

Table A3-5: Security Functional Requirements of Configurations

Finally, Table A3-6 summarizes the roles associated with each configuration:

Configuration	*Roles*
Java Card System - Closed Configuration	Java Card RE, Java Card VM, applets (RMIG)
Java Card System - Open Configuration	Java Card RE, Java Card VM, Installer, applet deletion manager, applets (RMIG), role for CarG functionalities (non specified)

Table A3-6: Configurations and Roles

APPENDIX 2: GLOSSARY

Term	Definition
AID	Application identifier, an ISO-7816 data format used for unique identification of Java Card applets (and certain kinds of files in card file systems). The Java Card platform uses the AID data format to identify applets and packages. AIDs are administered by the International Opens 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 namespace.
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 ([JCAPI22]). APDUs manage both the selection-cycle of the applets (through Java Card RE 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 Java Card RE owns an APDU object (which is a Java Card RE 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 any Java Card technology-based 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 ager	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. Unless explicitly said otherwise, in this document, CAD covers PCD.
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 Java Card RE 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 Java Card RE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command from the CAD or PCD with this applet's AID, the Java Card RE makes this applet the currently selected applet over the I/O interface that received the command. The Java Card RE sends all further APDU commands received over each interface to the currently selected applet on

	this interface ([JCRE22], Glossary).
Default applet	The applet that is selected after a card reset or upon completion of the PICC activation sequence on the contactless interface ([JCRE22], §4.1).
DPA	Differential Power Analysis is a form of side channel attack in which an attacker studies the power consumption of a cryptographic hardware device such as a smart card.
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 Java Card RE 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 Java Card VM 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 ([JCRE22],§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).
Java Card RE	The runtime environment under which Java programs in a smart card are executed. It is in charge of all the management features such as applet lifetime, applet isolation, object sharing, applet loading, applet initializing, transient objects, the transaction mechanism and so on.
Java Card RE Entry Point	An object owned by the Java Card RE context but accessible by any application. These methods are the gateways through which applets request privileged Java Card RE services: the instance methods associated to those objects may be invoked from any context, and when that occurs, a context switch to the Java Card RE context is performed. There are two categories of Java Card RE Entry Point Objects: Temporary ones and Permanent ones. As part of the firewall functionality, the Java Card RE detects and restricts attempts to

	store references to these objects.
Java Card RMI	Java Card Remote Method Invocation is the Java Card System version 2.2 and 3 Classic Edition 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 and that in Java Card System, version 3 Classic Edition, this functionality is optional.
Java Card System	Java Card System includes the Java Card RE, the Java Card VM, the Java Card API and the installer.
Java Card VM	The embedded interpreter of bytecodes. The Java Card VM 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 and 3 Classic Edition, that enables the opening of 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. Java Card platform, version 2.2.2 and 3 Classic Edition, enables opening up to twenty logical channels over each I/O interface (contacted or contactless).
NVRAM	Non-Volatile Random Access Memory, a type of memory that retains its contents when power is turned off.
Object deletion	The Java Card System version 2.2 and 3 Classic Edition 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 namespace 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.
PCD	Proximity Coupling Device. The PCD is a contactless card reader device.
PICC	Proximity Card. The PICC is a card with contactless capabilities.
RAM	Random Access Memory, is a type of computer memory that can be accessed randomly
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.
SWP	The Single Wire Protocol is a specification for a single-wire connection between the SIM card and a Near Field Communication (NFC) chip in a mobile handset
Transient object	An object whose contents are 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 Java Card RE. That also covers the packages. The associated subject(s), if applicable, is (are) an object(s) belonging to the javacard.framework.applet class.

Index

End of Document