Java Card™
Protection Profile Collection

Version 1.1

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054

May 2006

Legal Notice

Sun, Sun Microsystems, the Sun logo, Java, and Java Card are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DOCUMENTATION IS PROVIDED « AS IS » AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Sun, Sun Microsystems, le logo Sun, Java, et Java Card sont des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

LA DOCUMENTATION EST FOURNIE « EN L'ETAT » ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DAN LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

This document has been prepared by:

Trusted Logic SA
5, rue du Bailliage
78000 Versailles, France
http://www.trusted-logic.com

on behalf of Sun Microsystems, Inc.

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

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 USA
http://www.sun.com
JC_PP_feedback@sun.com

Secrétariat Général de la Défense Nationale
Direction Centrale de la Sécurité des Systèmes d'Information (DCSSI)
51, boulevard de Latour-Maubourg
75700 Paris 07 SP, France
http://www.ssi.gouv.fr/fr/dcssi
certification.dcssi@sgdn.pm.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 system, compliant with 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 that is put forward in this document is the definition of **groups** of requirements for each of the Java Card platform features proposed in the specifications. A particular choice of groups and a particular environment interacting with a Java Card platform give rise to a **configuration**, and then to the definition of the corresponding Protection Profile. Each of those Protection Profiles corresponds to a particular combination of features provided by a Java Card platform and the corresponding security architecture.

Four Protection Profiles, which define four specific configurations, are presented in this document. Two of them were chosen because they correspond to standard use-cases. The other two cover the largest range of features proposed in the latest version of Java Card technology, the difference being in the way verification of loaded applications is performed. The use of groups enables a modular construction of each configuration, enhancing the possibility of re-using large parts of it for the evaluations of other configurations, and simplifying its evolution across the future versions

of the Java Card technology. A special section with a comprehensive presentation of each configuration is also included as part of this document.

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

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

CONTENTS

1 Introduction ... 10
1.1 IDENTIFICATION ... 10
1.1.1 IDENTIFICATION OF THE DOCUMENT ... 10
1.1.2 ON THE CONFORMANCE OF SECURITY TARGETS ... 10
1.1.3 IDENTIFICATION OF THE PROTECTION PROFILES ... 11
1.2 REVISIONS AND COMMENTS ... 14
1.3 OVERVIEW ... 14
1.4 CC CONFORMANCE ... 15
1.5 TYPOGRAPHIC CONVENTIONS ... 16
1.6 ASSOCIATED DOCUMENTS ... 16
1.6.1 REFERENCE DOCUMENTS ... 16
1.6.2 RELATED DOCUMENTS ... 17
1.7 CONFIGURATIONS AND GROUPS ... 18
1.7.1 WHAT IS A GROUP? ... 18
1.7.2 WHAT IS A CONFIGURATION? ... 19
1.7.3 DEFINITION AND COMPOSITION OF GROUPS ... 19
2 TOE Description ... 21
2.1 PRODUCT TYPE ... 21
2.1.1 BYTECODE VERIFICATION ... 23
2.1.2 INSTALLATION OF APPLETS ... 23
2.1.3 THE CARD MANAGER (CM) ... 24
2.1.4 SMART CARD PLATFORM: OPERATING SYSTEM + CHIP + DEDICATED SOFTWARE 24
2.1.5 NATIVE APPLICATIONS ... 24
2.2 JAVA CARD 2.2 TECHNOLOGY ... 25
2.3 FUNCTIONAL COMPONENTS AND CONFIGURATIONS ... 26
2.3.1 CONFIGURATIONS ... 27
2.4 LIMITS OF THE TOE ... 31
2.4.1 SCOPE OF EVALUATION ... 31
2.4.2 THE TOE IN THE LIFE CYCLE OF THE SMART CARD ... 34
2.5 TOE INTENDED USAGE ... 36
2.6 PRODUCT RATIONALE ... 38
3 TOE Security Environment ... 39
3.1 SECURITY ASPECTS ... 39
3.2 ASSETS ... 46
3.2.1 USER DATA ... 46
3.2.2 TSF DATA ... 47
3.3 USERS & SUBJECTS ... 48
3.4 ASSUMPTIONS ... 49
3.4.1 ALL CONFIGURATIONS ... 49
3.4.2 MINIMAL CONFIGURATION ... 49
3.4.3 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION ... 49
3.4.4 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION ... 50
3.4.5 DEFENSIVE CONFIGURATION ... 50
3.5 THREATS ... 50
3.5.1 ALL CONFIGURATIONS ... 50
3.5.2 MINIMAL CONFIGURATION ... 52
3.5.3 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION ... 52
3.5.4 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION ... 53
3.5.5 DEFENSIVE CONFIGURATION ... 54
3.6 ORGANIZATIONAL SECURITY POLICIES ... 54
3.6.1 MINIMAL CONFIGURATION ... 54
3.6.2 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION ... 54
3.6.3 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION ... 54

3.6.4 DEFENSIVE CONFIGURATION 55
4 Security Objectives 56
4.1 SECURITY OBJECTIVES FOR THE TOE 56
4.1.1 ALL CONFIGURATIONS 56
4.1.2 MINIMAL CONFIGURATION 58
4.1.3 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION 58
4.1.4 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION 58
4.1.5 DEFENSIVE CONFIGURATION 59
4.2 SECURITY OBJECTIVES FOR THE ENVIRONMENT 59
4.2.1 ALL CONFIGURATIONS 59
4.2.2 MINIMAL CONFIGURATION 60
4.2.3 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION 61
4.2.4 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION 61
4.2.5 DEFENSIVE CONFIGURATION 61
5 IT Security Requirements 62
5.1 TOE AND IT ENVIRONMENT SECURITY REQUIREMENTS 62
5.1.1 COREG SECURITY FUNCTIONAL REQUIREMENTS 63
5.1.2 INSTG SECURITY FUNCTIONAL REQUIREMENTS 81
5.1.3 BCVG SECURITY FUNCTIONAL REQUIREMENTS 84
5.1.4 ADELG SECURITY FUNCTIONAL REQUIREMENTS 92
5.1.5 RMIG SECURITY FUNCTIONAL REQUIREMENTS 98
5.1.6 LCG SECURITY FUNCTIONAL REQUIREMENTS 104
5.1.7 ODELG SECURITY FUNCTIONAL REQUIREMENTS 107
5.1.8 CARG SECURITY FUNCTIONAL REQUIREMENTS 108
5.1.9 SCPG SECURITY FUNCTIONAL REQUIREMENTS 113
5.1.10 CMGRG SECURITY FUNCTIONAL REQUIREMENTS 115
5.1.11 EMG SECURITY FUNCTIONAL REQUIREMENTS 116
5.2 TOE SECURITY ASSURANCE REQUIREMENTS 119
6 Rationale 122
6.1 SECURITY OBJECTIVES RATIONALE 122
6.1.1 MINIMAL CONFIGURATION 122
6.1.2 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION 126
6.1.3 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION 132
6.1.4 DEFENSIVE CONFIGURATION 138
6.2 SECURITY REQUIREMENTS RATIONALE 143
6.2.1 MINIMAL CONFIGURATION 143
6.2.2 JAVA CARD SYSTEM STANDARD 2.1.1 CONFIGURATION 153
6.2.3 JAVA CARD SYSTEM STANDARD 2.2 CONFIGURATION 164
6.2.4 DEFENSIVE CONFIGURATION 177
7 Appendix: A Unified View of Configurations 189
8 Appendix: Glossary 195

LIST OF FIGURES

Figure 1: Usage environment of the TOE 22
Figure 2: TOE Limits for Minimal **configuration** 28
Figure 3: TOE Limits for Java Card System Standard 2.1.1 **configuration** 29
Figure 4: TOE Limits for Java Card System Standard 2.2 **configuration** 30
Figure 5: TOE Limits for Defensive **configuration** 30
Figure 6: Mandatory and optional components of the TOE 33
Figure 7: Smart Card Product Life Cycle 34

LIST OF TABLES

Table 1: Relationship between Groups and Configurations 32
Table 2: Relationship between Groups and Configurations 62
Table 3: Minimal **Configuration threats rationale** 125
Table 4: Minimal **Configuration assumptions rationale** 126
Table 5: Java Card System Standard 2.1.1 **Configuration threats rationale** 130
Table 6: Java Card System Standard 2.1.1 **Configuration assumptions rationale** 130
Table 7: Java Card System Standard 2.2 **Configuration threats rationale** 136
Table 8: Java Card System Standard 2.2 **Configuration assumptions rationale** 137
Table 9: Defensive **Configuration threats rationale** 141
Table 10: Defensive **Configuration assumptions rationale** 142
Table 11: Security requirements rationale for the Minimal **Configuration** 146
Table 12: Security requirements rationale for the group *SCPG* 147
Table 13: Functional Requirement Dependencies (Minimal**)** 149
Table 14: Security requirements rationale for the Java Card System Standard 2.1.1 **Configuration** 156
Table 15: Security requirements rationale for the group *SCPG* 157
Table 16: Functional Requirement Dependencies (Java Card System Standard 2.1.1**)** 160
Table 17: Security requirements rationale for the Java Card System Standard 2.2 **Configuration** ... 169
Table 18: Security requirements rationale for the group *SCPG* 170
Table 19: Functional Requirement Dependencies (Java Card System Standard 2.2**)** 173
Table 20: Security requirements rationale for the Defensive **Configuration** 182
Table 21: Security requirements rationale for the group *SCPG* 183
Table 22: Functional Requirement Dependencies (Defensive**)** 186
Table 23: Assumptions of Configurations 189
Table 24: Threats of Configurations 190
Table 25: TOE Security Objectives of Configurations 191
Table 26: Security objectives for the environment of Configurations 191
Table 27: Security Functional Requirements of Configurations 193
Table 28: Configurations and Roles 194

1 INTRODUCTION

This chapter identifies the document and the references it cites, presents its general structure, and introduces some key notions and notation conventions to be used in the following chapters. In addition to that, this chapter also gives the precise identification of the Protection Profiles that it embodies.

1.1 IDENTIFICATION

1.1.1 Identification of the Document

Author: Trusted Logic on behalf of Sun Microsystems, Inc.

Title: Java Card Protection Profile Collection

Version: 1.1, May 2006

1.1.2 On the Conformance of Security Targets

A security target that claims conformance with one of the Protection Profiles defined in this Java Card Protection Profile Collection document must define a compliant TOE and a security environement and include a unique identification of all their components (underlying smart card platform, bytecode verifier, card manager, native code, etc.). The TOE of such security target may coincide with or enrich the Java Card Protection Profile's TOE (actual TOEs will consist of the Java Card platform, possibly including native API, together with the chip component, the Dedicated Software and potentially native applications).

In those cases where the TOE defined in the security target is larger than that of the claimed Java Card Protection Profile, it is possible that some of Java Card Protection Profile's assumptions as well as objectives and security functional requirements for the environment need to be discarded to the benefit of new assumptions, objectives and security functional requirements, for the environment and for TOE itself. The security target conformance rationale must provide evidence that the new elements do not weaken the security with respect to the original Java Card Protection Profilestatements.

The case of the native APIs, excluded from the scope of the Java Card Protection Profile, is paradigmatic. For those TOEs that do not embody them, the product developer must provide envidence that the native APIs do fulfil the Java Card Protection Profile assumptions (cf. *A.NATIVE*), in other words, that native APIs do not violate the security policies stated fthe TOE. On the contrary, those TOEs that embody native APIs will probably discard this Java Card Protection Profile assumption and add specific objectives and security requirements for the TOE itself. In this later case, the product developer must provide full Common Criteria evidence that the native APIs satisfy all these new requierements.

1.1.3 Identification of the Protection Profiles

This section identifies the four Protection Profiles contained in this document. Each Protection Profile is identified by its unique name and the sections of the document that are listed in the item **Protection Profile organization**.

1.1.3.1 Minimal Configuration Protection Profile

Author: Trusted Logic on behalf of Sun Microsystems, Inc.

Title: Java Card System - Minimal Configuration Protection Profile

Version: 1.1, May 2006

Registration number: PP/0303

Protection Profile organization:

Section 2 provides general purpose and ***TOE description***.

Security aspects, ***assets*** and the ***links between users and subjects*** are provided in §3.1, §3.2 and §3.3 respectively.

Assumptions are provided in §3.4.1 and §3.4.2 and ***threats*** in §3.5.1 and §3.5.2.

The ***TOE security objectives*** are to be found in §4.1.1, and the ***IT environment objectives*** in §4.2.1 and §4.2.2.

The ***TOE security requirements*** are those of the group *CoreG* (§5.1.1), and the ***IT environment security requirements*** are the ones defined in the groups *BCVG* (§5.1.3), *SCPG* (§5.1.9) and *CMGRG* (§5.1.10). The ***TOE security assurance requirements*** are to be found in §5.2.

The ***rationale for security objectives and threats*** is provided in §6.1.1.1, the ***relation between security objectives and assumptions*** in §6.1.1.2.

The ***security requirements rationales*** are provided in §6.2.1.1 and §6.2.1.2; and the ***SFRs dependencies analysis*** in §6.2.1.3. The ***rationales for strength of function level, assurance requirements and consistency and mutual support*** are to be found in §6.2.1.4, §6.2.1.5 and §6.2.1.6 respectively.

Keywords: Multi-application Smart Card**,** Java Card Technology, Virtual Machine, Secure Runtime Environment.

Address: Sun Microsystems, Inc.; 4150 Network Circle, Santa Clara, CA 95054 USA.

1.1.3.2 Java Card System Standard 2.1.1 Configuration Protection Profile

Authors: Trusted Logic on behalf of Sun Microsystems, Inc.

Title: Java Card System - Standard 2.1.1 Configuration Protection Profile

Version: 1.1, May 2006

Registration number: PP/0304

Protection Profile organization:

Section 2 provides general purpose and ***TOE description***.

Security aspects, ***assets*** and the ***links between users and subjects*** are provided in §3.1, §3.2 and §3.3 respectively.

Assumptions are provided in §3.4.1 and §3.4.3, ***threats*** in §3.5.1 and §3.5.3 and ***organizational security policies*** in §3.6.2.

The ***TOE security objectives*** are to be found in §4.1.1 and §4.1.2, and the ***IT environment objectives*** in §4.2.1 and §4.2.3.

The ***TOE security requirements*** are those of the group CoreG (§5.1.1), InstG (§5.1.2) and CarG (§5.1.8). The ***IT environment security requirements*** are the ones defined in the groups BCVG (§5.1.3), SCPG (§5.1.9) and CMGRG (§5.1.10). The ***TOE security assurance requirements*** are to be found in §5.2.

The ***rationale for security objectives and threats*** is provided in §6.1.2.1, the ***relation between security objectives and assumptions*** in §6.1.2.2 and the rationale for the organizational security policies in §6.1.2.3.

The ***security requirements rationales*** are provided in §6.2.2.1 and §6.2.2.2; and the ***SFRs dependencies analysis*** in §6.2.2.3. The ***rationales for strength of function level, assurance requirements and consistency and mutual support*** are to be found in §6.2.2.4, §6.2.2.5 and §6.2.2.6 respectively.

Keywords: Multi-application Smart Card**,** Java Card Technology, Virtual Machine, Secure Runtime Environment.

Address: Sun Microsystems, Inc.; 4150 Network Circle, Santa Clara CA 95054 USA.

1.1.3.3 Java Card System Standard 2.2 Configuration Protection Profile

Authors: Trusted Logic on behalf of Sun Microsystems, Inc.

Title: Java Card System - Standard 2.2 Configuration Protection Profile

Version: 1.1, May 2006

Registration number: PP/0305

Protection Profile organization:

Section 2 provides general purpose and ***TOE description***.

Security aspects, ***assets*** and the ***links between users and subjects*** are provided in §3.1, §3.2 and §3.3 respectively.

Assumptions are provided in §3.4.1 and §3.4.4, ***threats*** in §3.5.1, §3.5.3 and §3.5.4; and ***organizational security policies*** in §3.6.2.

The ***TOE security objectives*** are to be found in §4.1.1, §4.1.2 and §4.1.4, and the ***IT environment objectives*** in §4.2.1, §4.2.3 and §4.2.4.

The ***TOE security requirements*** are those of the group CoreG (§5.1.1), InstG (§5.1.2), ADELG (§5.1.4), RMIG (§5.1.5), LCG (§5.1.6), ODELG (§5.1.7), CarG (§5.1.8) and EMG(§5.1.11). The ***IT environment***

security requirements are the ones defined in the groups BCVG (§5.1.3), SCPG (§5.1.9) and CMGRG (§5.1.10). The ***TOE security assurance requirements*** are to be found in §5.2.

The ***rationale for security objectives and threats*** is provided in §6.1.3.1, the ***relation between security objectives and assumptions*** in §6.1.3.2 and the rationale for the organizational security policies in §6.1.3.3.

The ***security requirements rationales*** are provided in §6.2.3.1 and §6.2.3.2; and the ***SFRs dependencies analysis*** in §6.2.3.3. The ***rationales for strength of function level, assurance requirements and consistency and mutual support*** are to be found in §6.2.3.4, §6.2.3.5 and §6.2.3.6 respectively.

Keywords: Multi-application Smart Card**,** Java Card Technology, Virtual Machine, Secure Runtime Environment.

Address: Sun Microsystems, Inc.; 4150 Network Circle, CA 95054 USA.

1.1.3.4 Defensive Configuration Protection Profile

Authors: Trusted Logic on behalf of Sun Microsystems, Inc.

Title: Java Card System - Defensive Configuration Protection Profile

Version: 1.1, May 2006

Registration number: PP/0306

Protection Profile organization:

Section 2 provides general purpose and ***TOE description***.

Security aspects, ***assets*** and the ***links between users and subjects*** are provided in §3.1, §3.2 and §3.3 respectively.

Assumptions are provided in §3.4.1 and §3.4.5, ***threats*** in §3.5.1 and §3.5.5. There are no ***organizational security policies***.

The ***TOE security objectives*** are to be found in §4.1.1 and §4.1.5, and the ***IT environment objectives*** in §4.2.1 and §4.2.5.

The ***TOE security requirements*** are those of the group CoreG (§5.1.1), InstG (§5.1.2), BCVG (§5.1.3), ADELG (§5.1.4), RMIG (§5.1.5), LCG (§5.1.6), ODELG (§5.1.7) and EMG(§5.1.11). The ***IT environment security requirements*** are the ones defined in the groups SCPG (§5.1.9) and CMGRG (§5.1.10). The ***TOE security assurance requirements*** are to be found in §5.2.

The ***rationale for security objectives and threats*** is provided in §6.1.4.1, the ***relation between security objectives and assumptions*** in §6.1.4.2

The ***security requirements rationales*** are provided in §6.2.4.1 and §6.2.4.2; and the ***SFRs dependencies analysis*** in §6.2.4.3. The ***rationales for strength of function level, assurance requirements and consistency and mutual support*** are to be found in §6.2.4.4, §6.2.4.5 and §6.2.4.6 respectively.

Keywords: Multi-application Smart Card**,** Java Card Technology, Virtual Machine, Secure Runtime Environment.

Address: Sun Microsystems, Inc.; 4150 Network Circle, Santa Clara, CA 95054 USA.

1.2 REVISIONS AND COMMENTS

Version	Issue date	Comments
1.0	July 1999	First version without sharing and post-issuance downloading. Used in Vocable project.
1.1	July 2001	Java Card System Protection Profile. Based on Java Card System 2.1.1. Distribution to card issuers and operators for comments.
1.2	November 2001	Integration of comments coming from card issuers and operators. Available to licensees for comments on Sun Website.
2.0	May 2002	New structure of the protection profile in terms of configurations and groups of security requirements. Introduction of Java Card System 2.2 features (RMI, logical channels, applet deletion and object deletion).
2.1	October 2002	Integration of comments and remarks coming from Sun and from the Java Card Forum Security Task Force. First version submitted for evaluation.
0.1	January 2003	Java Card System Protection Profile Collection. Stand-alone (per configuration) rationales of Security Objectives and Security Functional Requirements.
0.2	February 2003	Revised to comply with the request of the evaluator.
1.0	June 2003	Revised to comply with the request of the evaluator.
1.0b	August 2003	Final version
1.1Nov05	November 2005	Update for Java Card Platform, version 2.2.1 and Java Card Platform, version 2.2.2
1.1Dec05	December 2005	Biometry and user remarks
1.1March06	March 2006	Editorial update
1.1May06	May 2006	Inclusion of all CC final interpretations applicable to CCv2.1 Updates according to Java Card Platform licensee comments.

1.3 OVERVIEW

The aim of this document is to describe a unified framework for the definition of a Protection Profile for a Java Card System compliant with *Sun Microsystems* Java Card specifications, versions 2.1.1, 2.2, 2.2.1 or 2.2.2. An important issue addressed by this document is the possibility of having different configurations for a Java Card platform, resulting from the optional features of Java Card technology and the security architectures for bytecode verification that have been conceived so far. Moreover, this document includes the definition of four Protection Profiles, one for each of the configurations that

have been considered. The Protection Profiles should provide a valuable input for the development of Java Card Platform security targets.

The main security goal of the Java Card platform is to counter the unauthorized disclosure or modification of the code and data of both the applications and its own, as well as of any other data that may be sensitive such as application software, keys, PINs, biometric templates, and so on.

In order to achieve these goals, the Java Card platform provides some security features. The most important are the following mechanisms:

- Logical separation of the data used by different applications (firewall)
- Static analysis of the code before installation (bytecode verification)
- Preservation of the code integrity between static verification and installation on the card.
- Use of security services for applications such as
  - Specific management of cryptographic keys and PIN
  - Cryptographic authentication and ciphering mechanisms

The structure of the document is very close to the standard one, specified in [CC1], for a Protection Profile. In addition to the usual sections, the following special ones are also included:

Section §1.7 explains how this document addresses the possibility of having different configurations. It presents the general notions of *group of requirements* and *configuration*, which constitute the basis for the definition and evaluation of use-cases.

After the general description of all the features offered by a Java Card System, Section §2.3.1 introduces the ten groups of requirements induced by those features, and the four particular configurations proposed in this document, called Minimal, Java Card System Standard 2.1.1, Java Card System Standard 2.2 and Defensive. When the definition of a CC component depends on the configuration, the structure of the corresponding section introduces a new level of sub-sections, one for each of the configurations aforementioned. For instance, the section *Security Objectives for the TOE* is divided into four sub-sections, one containing the objectives of the Minimal configuration, one with those objectives that are specific to the Java Card System Standard 2.1.1 configuration, one with those objectives that are specific to the Java Card System Standard 2.2 configuration and a fourth one with those specific to the Defensive configuration.

Two appendices are also included:

Appendix §7 provides a unified view of the configurations defined in the document.

Appendix §8 contains a glossary of technical terms used in the document

1.4 CC CONFORMANCE

This document contains four Protection Profiles.

The Protection Profiles have been built with Common Criteria (CC) Version 2.1 (ISO/IEC 15408 Evaluation Criteria for Information Technology Security; Part 1: Introduction and general model, Part 2: Security functional requirements, and Part 3: Security assurance requirements) and Common Methodology for Information Technology Security Evaluation (CEM-97/017, Part 1: Introduction and General Model, Version 0.6, 97/01/11 and CEM-99/045, Part 2: Evaluation Methodology, Version 1.0,August 1999). All final interpretations applicable to CC 2.1 have been incorporated, in particular: RI#65, RI#103, RI#104, RI#201, RI#220, RI#228 and RI#232.

Each Protection Profile is Part 2 and Part 3 conformant.

The assurance requirement of each Protection Profile is **EAL 4 augmented**. Augmentation results from the selection of:

AVA_VLA.3 Vulnerability Assessment - Vulnerability Analysis - Moderately resistant, and
ADV_IMP.2 Development – Implementation Representation – Implementation of the TSF.

The minimum strength of function level of each Protection Profile is **SOF-medium**.

1.5 TYPOGRAPHIC CONVENTIONS

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

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

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

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

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

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

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

1.6 ASSOCIATED DOCUMENTS

1.6.1 Reference Documents

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

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

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

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

[JCVM21] *Java Card 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, version 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*. March 2006. 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.

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

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

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

1.6.2 Related Documents

The following list is in no way exhaustive.

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

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

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

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

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

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

1.7 CONFIGURATIONS AND GROUPS

The Java Card System is a generic platform that can be used in numerous applications. Smart Card products have different needs depending, for instance, whether it is a banking card or a pay-TV one. To retain a high level of flexibility this document introduces the notions of *group* and *configuration*.

1.7.1 What is a Group?

CC packages are "A reusable set of either functional or assurance components, combined together to satisfy a set of identified security objectives" [CC1]. Practically, however, it is common for security functions to be grouped into functional modules, a fact that is acknowledged by the CC (see for instance the ADV_FSP and ADV_HLD assurance classes). These modules are usually associated to specific security aspects that contribute to meet a precise requirement, which in turn induces a similar division of the Security Functional Requirements ("SFR"). The **groups** put forward in this document then are **sets of identified security requirements**, and they are similar to CC packages.

The association between the objectives and the SFRs, however, is looser than required by the CC evaluation: the SFRs in a group may not be able to completely meet the objectives to which they are associated. This is similar to the case where an objective is met by a combination of SFRs for the TOE and SARs (Security Assurance Requirements), which apply to the environment: a group only *contributes* to meet an objective, and may not be sufficient alone. For instance, an access control policy (FDP_ACF.1) may belong to one group, while the initialization of its related security attributes (FMT_MSA.3) belongs to another group.

Also, one can consider groups solely as a way to structure the SFRs for a better understanding.

This document introduces, among others, groups of requirements concerning bytecode verification, installation and deletion of applications, transmission of applications to the card and isolation of application data during execution.

1.7.2 What is a Configuration?

Configurations correspond to the use-cases to be evaluated. Such use-cases arise from the choice of the different optional features proposed by the Java Card technology (like post-issuance application downloading, 2.2 version features), and the different security architectures that have been conceived so far for this technology (off-card verification or on-card verification).

Each configuration has its own security objectives, which determine the groups of requirements to be chosen in order to meet those objectives. Moreover, even if configurations may have the same global collection of objectives, some of them may be objectives for the TOE in one configuration and objectives of the environment in another one. A configuration is hence described setting up the precise limits of the TOE, a definite environment for it, and the groups of requirements to be used (notice that groups, as packages, do *not* contain any environmental description). Thereby, an ad-hoc rationale has to be developed for each configuration too.

From a different perspective, we may also see a configuration as a consistent and complete set of groups (in an environment) that is suitable for an evaluation and certification.

This document introduces four configurations and the corresponding Protection Profiles to be evaluated. They are defined in the next chapter.

1.7.3 Definition and Composition of Groups

This section contains some remarks regarding the composition of security requirements into groups, as well as how those groups can be assembled together in a consistent way.

The Common Criteria scheme defines several formal operations that can be applied to a functional component: iteration for repeated use, selection, assignment and refinement ([CC1], §2.1.4). The classification of the SFRs considered in this document into separate groups sometimes led to unpleasant repetitions. For instance, the ***FMT_SMR.1*** component, which defines the known security roles for the TOE, should essentially appear once in a security target, but the actual set of security roles to be considered depends on the configuration. In the same vein, the ***FMT_MSA.1*** component is repeated in each of the groups that introduce security attributes, although there is no obvious reason to iterate it, as it has no applicable selection or assignment operation. On the other hand, each group defines a role that is only meaningful when it is included in the considered configuration, so repeating it for each group provides a more accurate definition of the group of requirements.

Whereas the choice has been made to *repeat* the component within each group, the reader shall not understand such repetition as *iteration* in the formal CC sense, but shall consider these as a unique instance. Thus, each configuration really contains one ***FMT_SMR.1*** component, whose list of roles is given by all the roles appearing in the groups of the configuration.

A similar issue is raised by the components where a security policy (for access control or information flow) has to be assigned or selected in the component. For instance, the component ***FMT_MSA.1*** restricts the privileges granted to a given role with regard to the security attributes of a given policy. However, it could be the case that two security functions, one defined in a group, G1, and the other defined in another group, G2, make both use of a security attribute that is common to two policies, SP1 and SP2. Moreover, the possibility of modifying the shared security attribute may be restricted in G1 to the role R1 and in G2 to another role R2. Then, in those configurations including both the groups G1 and G2, it shall be understood that the modification of the shared attribute is actually restricted to <u>both</u> R1 and R2 by the enforcement of the policies SP1 and SP2. As no such operation of component composition is specified in the Common Criteria, and to prevent any possible misunderstanding, an application note is added to the component of the first group (G1) notifying the ST author that the list of roles enabled to modify the attribute actually depends on the configuration, and could be potentially extended by the inclusion of other groups.

Finally, the security policies included in certain groups of requirements should actually be understood as a complement to other security policies, in the sense that they extend them with new access control or information flow rules. This is the case, for instance, of the logical channel group, which extends the firewall access control policy with new rules concerning logical channels.

2 TOE DESCRIPTION

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

2.1 PRODUCT TYPE

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 [JCVM21]. The Java Card platform is a smart card platform enabled with Java Card technology. This technology allows for multiple applications to run on a single card and provides facilities for secure interoperability of applications. Applications for the Java Card platform are referred to as Java Card technology-based applets ("Java Card applets" or "applets") .

The version 2.1.1 of the Java Card platform is specified in [JCVM21], [JCRE21] and [JCAPI21]. It consists of the Java Card platform virtual machine ("*Java Card virtual machine" or "Java Card VM*"), the Java Card platform runtime environment ("Java Card runtime environment" or "*Java Card RE"RE*) and the Java Card Application Programming Interface (API).

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

The *Java Card VM* is essentially an abstract machine that specifies the behavior of the bytecode interpreter to be embedded in the card. The *Java Card RE* is responsible for card resource management, communication, *applet* execution, and on-card system and *applet* security. The API provides classes and interfaces for the core functionality of a Java Card applet. 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. The Java Card API is compatible with formal international standards, such as ISO7816, and industry specific standards, such as EMV (Europay/Master Card/Visa).

In certain use-cases, *applet*s can be loaded and installed on a Java Card platform after the card has been issued. This provides, for instance, card issuers with the ability to dynamically respond to their customer's changing needs. For example, if a customer decides to change the frequent flyer program associated with the card, the card issuer can make this change, without having to issue a new card. Moreover, *applet*s from different vendors can coexist in a single card, and they can even share information. An *applet*, however, is usually intended to store highly sensitive information, so the sharing of that information must be carefully limited. In the Java Card platform, *applet* isolation is achieved through the *applet firewall* mechanism ([JCRE21][JCRE22], §6.1). That mechanism confines an *applet* to its own designated memory area, thus each *applet* is prevented from accessing fields and operations of objects owned by other *applet*s, unless an interface is explicitly provided (by the *applet* who owns it) for allowing access to that information. *Java Card VM* dynamically enforces the *firewall*, that is, at runtime. However *applet* isolation cannot entirely be granted by the *firewall* mechanism if certain integrity conditions are not satisfied by the applications loaded on the card. Those conditions can be statically verified to hold by a bytecode verifier.

Figure 1 replaces the different components of the *Java Card System* in their environment. The development of the application, as well as the compilation and conversion steps (see below), is not included in the usage environment of the TOE.

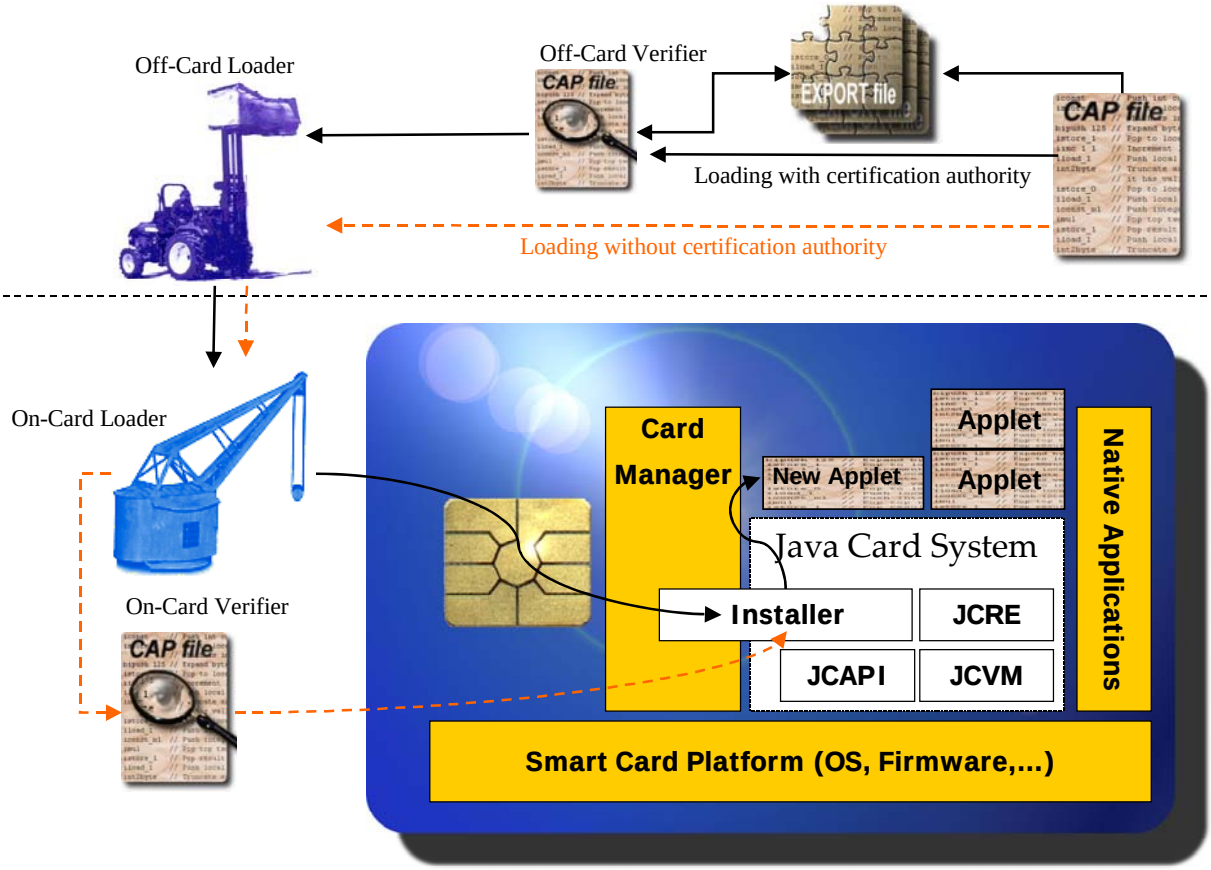

Figure 1: Usage environment of the TOE

One of the several possible scenarios depicted by Figure 1, concerning the development, loading and execution lifetime of an *applet*, is described in what follows. The chosen use-case involves almost all of the TOE and IT environment components considered in this document:

The development of the source code of the *applet* is carried on in a Java programming environment. The compilation of that code will then produce the corresponding *class* file. Then, this latter file is processed by the **converter**[1], which, on the one hand, validates the code and generates a converted applet (*CAP*) file, the equivalent of a Java™ class file for the Java Card platform. A *CAP file* contains an executable binary representation of the classes of a *package*. A *package* is a name space within the Java programming language that may contain *class*es and *interface*s, and in the context of Java Card technology, it defines either a user library, or one or several *applets.* Then, the (off-card) **bytecode verifier** checks the integrity of the *CAP file*. After the validation is carried out, the *CAP file* is then loaded into the card making use of a safe **loading** mechanism. Once loaded into the card the file is **linked**, what makes it possible in turn to **install**, if defined, instances of any of the *applet*s defined in the file. During the installation process the *applet* is registered on the card by using an application identifier (*AID*). This *AID* will allow the identification of unique *applet* instances within the card. In

[1] The converter is defined in the specifications as the off-card component of the *Java Card virtual Machine*.

particular, the AID is used for selecting the applet instance for execution. The on-card bytecode interpreter performs the execution of the *applet*'s code.

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

A brief description of some of the new features introduced in versions 2.2, 2.2.1 and 2.2.2 of the Java Card platform is also included.

2.1.1 Bytecode Verification

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

2.1.2 Installation of applets

The *installer* is the part of the on-card component of the platform dealing with downloading, linking and installation of new *package*s, as described in [JCRE21]. Once selected, it receives the *CAP file*, stores the classes of the *package* on the card, initializes static data, if any, and installs any applets contained in the package.

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

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

LOADING

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

LINKING

The linking process consists of a rearrangement of the information contained in the *CAP file* in order to speed up the execution of the applications. There is a first step where indirect external and internal references contained in the file are resolved by replacing those references with direct ones. This is what is referred to as the *resolution* step. In the next step, called in [JVM] the *preparation* step, the static field image[2] and the statically initialized arrays defined in the file are allocated. Those arrays in turn are also initialized, thus giving rise to what shall constitute the initial state of the *package* for the embedded interpreter.

2.1.3 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* (see below, §2.4.2.2). The card manager is in charge of the life cycle of the whole card, as well as the installed applications (*applet*s). It may have other roles (such as the management of security domains and enforcement of the card issuer security policies) that we do not detail here, as they are not in the scope of the TOE and are implementation–dependent.

The card manager's role is also to manage and control the communication between the card and the card acceptance device (*CAD*) or the proximity–coupling device (PCD)[3]. It is the controller of the card, but relies on the TOE to manage the runtime of client *applet*s. On the other hand, the TOE relies on the card manager for some of its security functions (§2.4.2.2).

A candidate for this component is the Global Platform card manager ([GP]).

2.1.4 Smart Card Platform: Operating System + Chip + Dedicated Software

The smart card platform (*SCP*) is composed of a micro-controller and an operating system. It provides memory management functions (such as separate interface to RAM and NVRAM), I/O functions that are compliant with ISO standards, transaction facilities, and secure (shielded, native) implementation of cryptographic functions. It also contains dedicated software (DS), which provides an interface with the integrated circuit (IC).

Finally, it is likely that the *SCP* has to be evaluated along with the TOE in order to claim a good level of assurance, when needed.

2.1.5 Native Applications

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

---

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

[3] 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.

It is obvious that such native code presents a threat to the security of the TOE and to user *applets*. Therefore, the Protection Profiles will require for native applications to be conformant with the TOE so as to ensure that they do not provide a means to circumvent or jeopardize the TSFs.

2.2 JAVA CARD 2.2 TECHNOLOGY

This document is also concerned with the new features included in the Java Card System 2.2 platform specification ([JCVM22], [JCRE22], [JCAPI22], [JCVM222], [JCRE222], [JCAPI221], [JCAPI222], [JCVM221], [JCVM222]), namely, the support of logical channels, *applet* and *package* deletion, object deletion, remote method invocation, external memory facilities and contactless I/O interface

Any of the four components described below, when included in a configuration, is to be considered as part of the*Java Card System*.

JAVA CARD REMOTE METHOD INVOCATION (JAVA CARD RMI)

Java Card platform Remote Method Invocation (*Java Card RMI*) 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 Java Card RMI facilities are introduced as part of an extended framework aimed at improving the productivity of application developers for the Java Card platform, on the one hand, promoting a technology that is used today in many client-server applications, and, on the other hand, freeing the task of application developing of having to deal directly with the card-specific programming model. Moreover, *Java Card RMI* enables the use of the Java technology for both the card and the terminal.

The applet owner of those objects controls the access to exported objects and the *Java Card RE* ensures coherence and synchronization of the remote object with its on-card representative.

APPLET DELETION MANAGER (ADEL)

The *applet deletion manager* is the on-card component that embodies the mechanisms necessary to delete an *applet* on smart cards using Java Card technology. If the implementation of the *Java Card System* includes a post-issuance *installer*, then an *applet deletion manager* that supports the behavior specified in [JCRE22],§11.3, is also required. The *applet deletion manager* must appear as an *applet* to the *CAD*. Therefore, it has an *AID*, and it must be selected for execution. There are three categories of applet deletion requirements in Java Card System, version 2.2 ([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 *Java Card RE* management structures.

Deletion of an *applet package* and contained instances, which is the removal of the card resident code and *Java Card RE* structures associated with the *applet package*, and all the *applet* instances in the context of the *package*.

LOGICAL CHANNELS

The Java Card platform, version 2.2, provides support for *logical channel*s, that is, the ability to allow a terminal to open up multiple sessions into the smart card for each I/O interface (contacted and

contactless), one session per *logical channel* ([JCRE22],§4 and [JCRE222],§4). Commands may be issued on a logical channel over an I/O interface to instruct the card either to open or to close a logical channel on the same I/O interface. 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 channel*s at the same time, or accepting other *applet*s belonging to the same *package* to be selected simultaneously. These *applet*s are referred to as *multiselectable*. A *non-multiselectable applet* can be active at most on one channel. *Applet*s within a *package* are either all multiselectable or all non-multiselectable.

OBJECT DELETION

The Java Card platform, version 2.2, offers an (optional) object deletion mechanism. This mechanism is requested by an *applet* instance, and the *Java Card RE* must ensure that any unreferenced object owned by that instance is deleted and the associated space must be recovered for reuse. Applications designed to run on a platform providing this facility can make use of it by invoking the method **`requestObjectDeletion()`**[JCAPI22].

EXTERNAL MEMORY

The Java Card platform, version 2.2.2, provides an API-based mechanism to access the external memory outside the addressable Java Card VM space. An *applet* instance reads from/writes to the external memory through the methods of the MemoryAccess interface. It is up to the implementation to ensure that no instance of this interface can ever be created or used to access memory that is directly accessed and managed by the Java Card RE for code, heap and other data structures [JCAPI222].

2.3 FUNCTIONAL COMPONENTS AND CONFIGURATIONS

In §1.7.1 the concept of group of SFRs was introduced and the role they play in the Protection Profiles is defined in this document. The following list describes the groups of security requirements that have been used in those Protection Profiles. The definition of each of those groups is strongly influenced by the behavior of the functional components described in the previous section:

SCP group	The *SCPG* contains the security requirements for the smart card platform, that is, operating system and chip that the Java Card System is implemented upon. It does not define requirements for the TOE but for its IT environment.
Core group	The *CoreG* contains the basic requirements concerning the runtime environment of the Java Card System, such as the firewall policy and the requirements related to the Java Card API.
Bytecode verification group	The *BCVG* contains the security requirements concerning the bytecode verification of the application code to be loaded on the card. This group of SFRs may apply to the TOE or to its IT environment depending on the configuration.
Installation group	The *InstG* contains the security requirements concerning the installation of post-issuance applications. It does not address card management issues in the broad sense, but only those security aspects of the installation procedure that are related to applet execution. Those aspects are described in [JCRE21]§11.1.5 *Installer behavior.*

Applet deletion group	The *ADELG* contains the security requirements for erasing installed applets from the card, a new feature introduced in Java Card System 2.2. It can also be used as a basis for any other application deletion requirements.
Remote Method Invocation (RMI) group	The *RMIG* contains the security requirements for the remote method invocation features, which provides a new protocol of communication between the terminal and the applets. This was introduced in Java Card System 2.2.
Logical channels group	The *LCG* contains the security requirements for the logical channels, which provide a runtime environment where several applets can be simultaneously selected or a single one can be selected more than once. This is a Java Card System 2.2 feature.
Object deletion group	The *ODELG* contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card System 2.2 feature.
Secure carrier group	The *CarG* group contains minimal requirements for secure downloading of applications on the card. This group contains the security requirements for preventing, in those configurations that do not support on-card static or dynamic bytecodesverification, the installation of a *package* that has not been bytecode verified, or that has been modified after bytecode verification.
Card manager group	The *CMGRG* contains the minimal requirements that allow defining a policy for controlling access to card content management operations and for expressing card issuer security concerns.
Extended Memory group	The *EMG* contains the requirements for a secure management of the external memory accessible to applet instances. This is a Java Card System 2.2.2 feature.

2.3.1 Configurations

The following are the configurations, among the several ones that can be defined, which are addressed in this document. They have been chosen either because they correspond to existing use-cases, or because they cover the largest range of features of the Java Card platform.

2.3.1.1 Minimal Configuration

The minimal configuration corresponds to a multi-application card where no downloading of post-issuance applications is allowed. The TOE is the simplest Java Card RE and its IT environment is the smart card platform, the bytecode verifier and the card manager. Only the groups *SCPG*, *CoreG*, *BCVG* and *CMGRG* are included in this configuration.

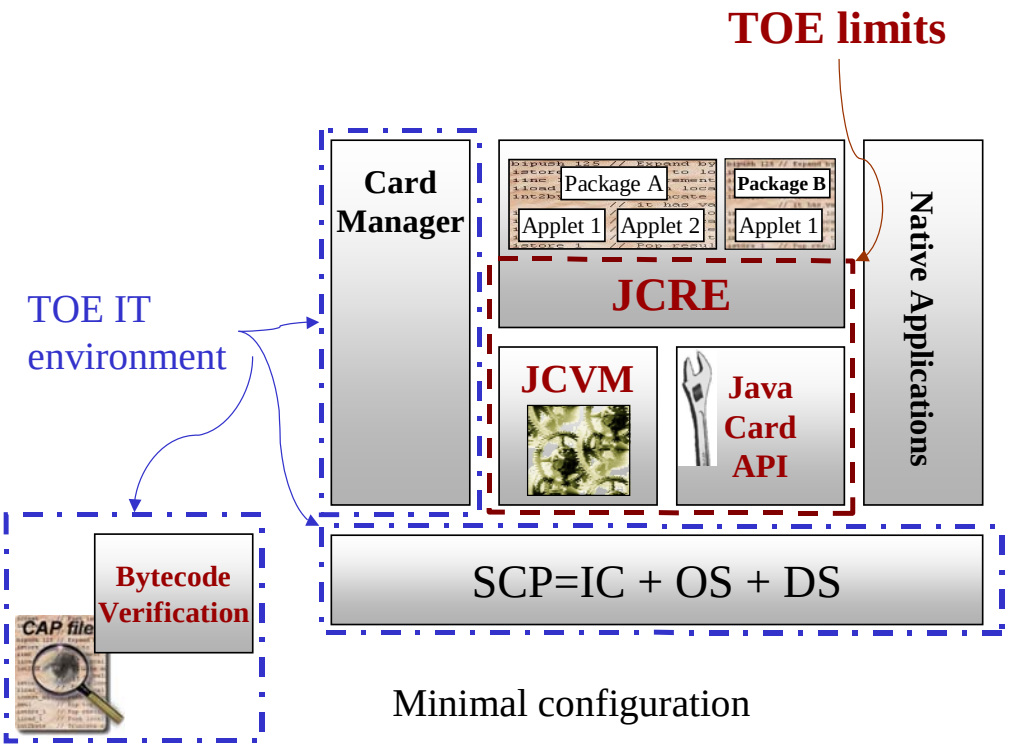

Figure 2: TOE Limits for Minimal configuration

2.3.1.2 Java Card System Standard 2.1.1 Configuration

The **Java Card System Standard 2.1.1** configuration corresponds to a platform that includes all the functionalities described in Java Card specifications, version 2.1.1. It extends the **Minimal** configuration with the security requirements for downloading post-issuance applications[4] that have been previously verified by an off-card remote trusted IT component. The loader and the installer form part of the TOE, and therefore the groups *CarG* and *InstG* are included. The **Java Card System Standard 2.1.1** configuration however does not provide functionalities for deletion of applets. Bytecode verification and card management applies to the TOE IT environment.

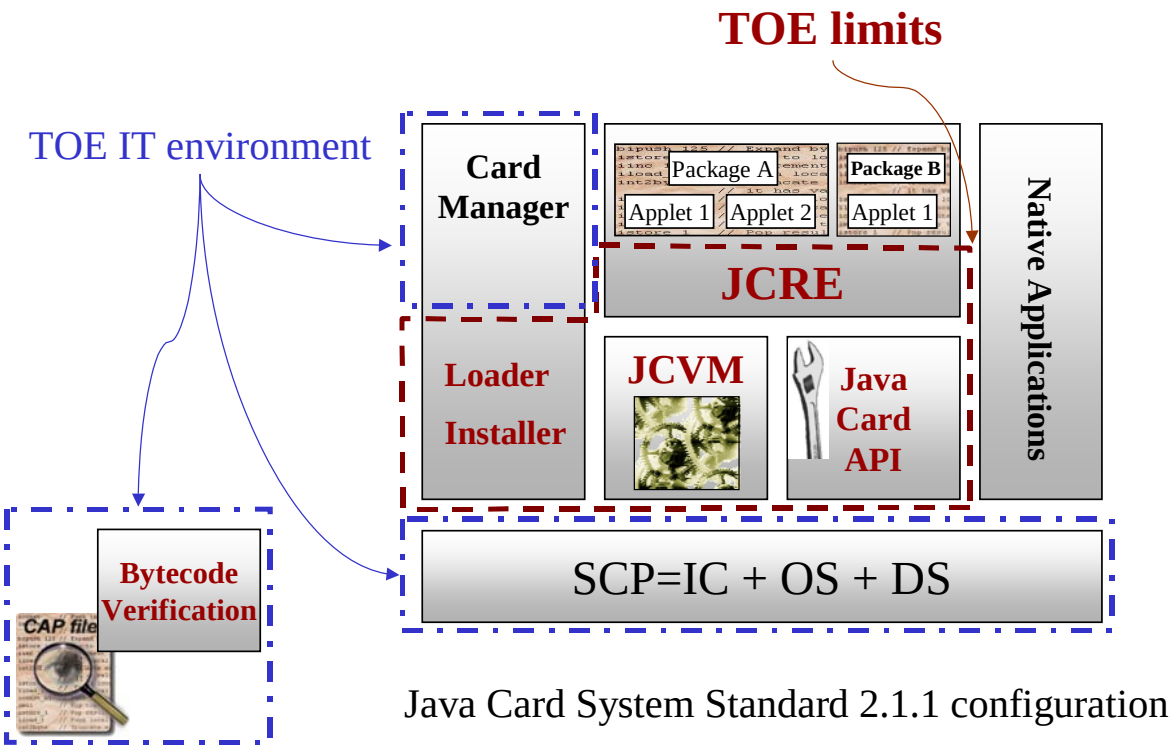

Figure 3: TOE Limits for Java Card System Standard 2.1.1 **configuration**

2.3.1.3 Java Card System Standard 2.2 Configuration

This configuration extends the **Java Card System Standard 2.1.1** configuration with all the features introduced in the Java Card specifications, version 2.2 (Java Card RMI, logical channels, applet deletion and object deletion). Therefore, the groups *CarG*, *InstG*, *RMIG*, *LCG*, *ADELG* and *ODELG* are included. Group *EMG* is also included, but it is only relevant if the TOE implements the External Memory API of the Java Card System 2.2.2. Bytecode verification and card management apply to the TOE IT environment.

[4] The applet Installer is an optional feature of Java Card System, version 2.1.1.

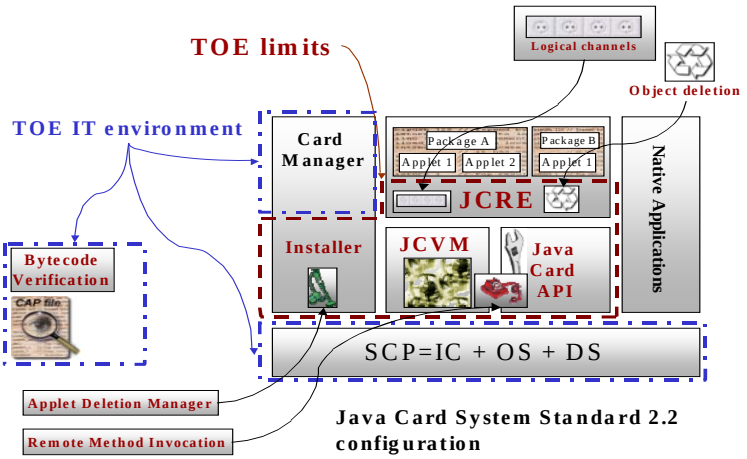

Figure 4: TOE Limits for Java Card System Standard 2.2 **configuration**

2.3.1.4 *Defensive Configuration*

This configuration, like the **Java Card System Standard 2.2** configuration, also includes all the features considered in version 2.2 of the Java Card specifications. In addition to that, bytecode verification is performed on-card and the bytecode verifier is then a component of the TOE. The *BCVG* group is therefore also included, not being the case of the group *CarG* since installation of malicious applets is prevented independently from the origin of the application and the way it has been downloaded on the card. Card management applies to the TOE IT environment.

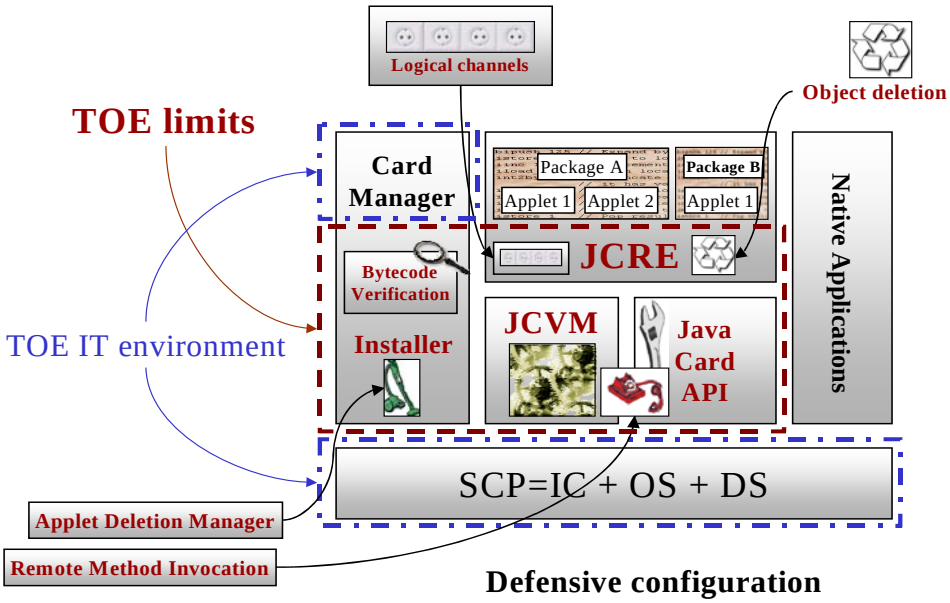

Figure 5: TOE Limits for Defensive **configuration**

2.4 LIMITS OF THE TOE

2.4.1 Scope of Evaluation

The scope of the TOE is the *Java Card System*. The integrated circuit, the operating system and the dedicated software of the smart card are not part of the TOE. Neither is part of the TOE any piece of native code that does not contribute to its implementation, like a native application embedded together with Java Card applets. However, the *Java Card System* is used by the *applet*s and interacts with the *SCP*, the card manager and other components of the smart card. All of them are thus part of the TOE IT environment, and are included in the scope of evaluation of the Protection Profiles.

Regarding the code of the TOE, one may distinguish the *Java Card System* as a "pure software component" from the actual product, which is the very same software running on a smart card, as a part of an ST (see §2.4.2). While the scope of the Protection Profiles does not include the development cycle of the smart card, the good working order of the TOE much depends on the way the TOE is handled during the manufacturing process of the card (for instance, how it is embedded into the card). Thus the scope of evaluation actually includes more than the TOE itself. The Common Criteria acknowledges this situation, allowing security requirements applying to the development and construction of the TOE, stated in several SARs (security assurance requirements), particularly those from the ADO (delivery) and ACM (configuration) classes [CC3].

Let us also remark that the code of the applets is not part of the code of the TOE, but just data managed by the TOE. Moreover, the scope of the Protection Profiles does not include all the stages in the development cycle of a Java Card applet described in §2.1. Applets are only considered in their CAP format, and the process of compiling the source code of an application and converting it into the CAP format does not regard the TOE or its environment. On the contrary, the process of verifying applications in its CAP format and loading it on the card is a crucial part of the TOE environment and plays an important role as a complement of the TSFs included in the configuration. The Protection Profiles assume that the loading of applications pre-issuance is made in a secure environment. For post-issuance phases, the card will need to protect itself so that *applet*s can only be loaded within a secured environment[5].

Native applications (see §2.1.5) may be placed into the card not through the *installer* component of the *Java Card System*, but by directly embedding them into the IC during the fabrication of the smart card, along with that of the *Java Card System*. This is the usual way to have native methods installed, but the process is not limited to them, and *applet*s and API *package*s may also be installed at a time where the TOE is not yet operational. This also advocates for including several security assurance requirements on the life cycle of the smart card, since native applications are not under the control of the *Java Card System.*

It is also important to notice that the actual definition of the *Java Card System* (and thus the limits of the TOE) varies in accordance with the configuration under consideration. Figure 6 illustrates the components that are always inside the perimeter of the *Java Card System*, and the different optional components that may be also included.

[5] This protection is likely to be on the behalf of the card manager.

2.4.1.1 Relationship between Configurations and Groups

The following table illustrates the relationship between the chosen configurations and the groups described in §2.3. For each configuration, if a group is included it can be either part of the TOE or part of the IT environment. This holds, for instance, for the bytecode verification: when not performed on-card, it is part of the IT environment.

Group (group name)	Minimal	Java Card System Standard 2.1.1	Java Card System Standard 2.2	Defensive
Core (*CoreG*)	TOE	TOE	TOE	TOE
Smart card platform (*SCPG*)	IT	IT	IT	IT
Installer (*InstG*)	--	TOE	TOE	TOE
RMI (*RMIG*)	--	--	TOE	TOE
Logical channels (*LCG*)	--	--	TOE	TOE
Object deletion (*ODELG*)	--	--	TOE	TOE
Bytecode verification (*BCVG*)	IT	IT	IT	TOE
Applet deletion (*ADELG*)	--	--	TOE	TOE
Secure carrier (*CarG*)	--	TOE	TOE	--
Card manager (*CMGRG*)	IT	IT	IT	IT
External Memory (*EMG*)	--	--	TOE	TOE

Table 1: Relationship between Groups and Configurations

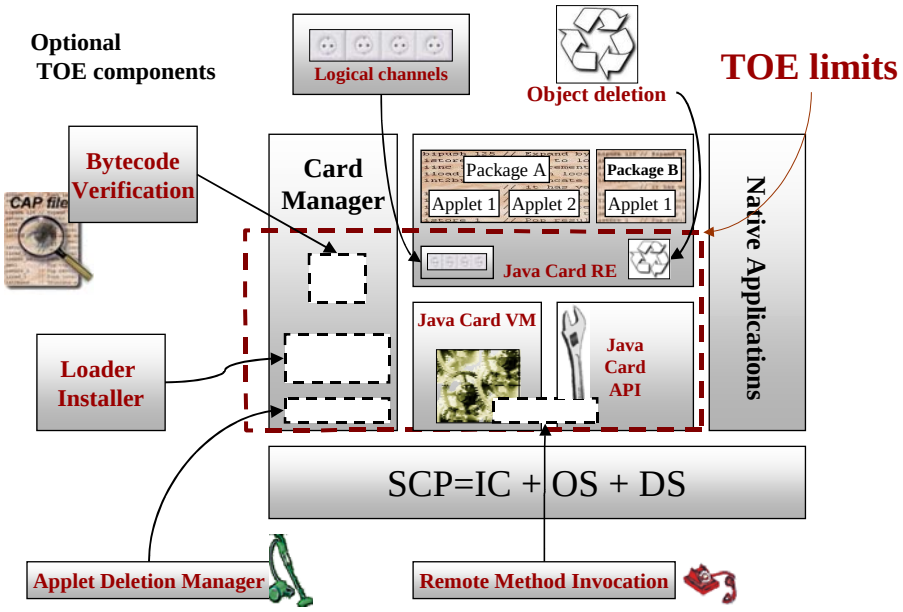

Figure 6: Mandatory and optional components of the TOE

2.4.2 The TOE in the Life Cycle of the Smart Card

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

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

The life cycle of the TOE, which is only a part of the smart card life cycle, can be reduced to the **three stages** pictured in Figure 7, called **Development**, **Production & Personalization**, and **Usage**.

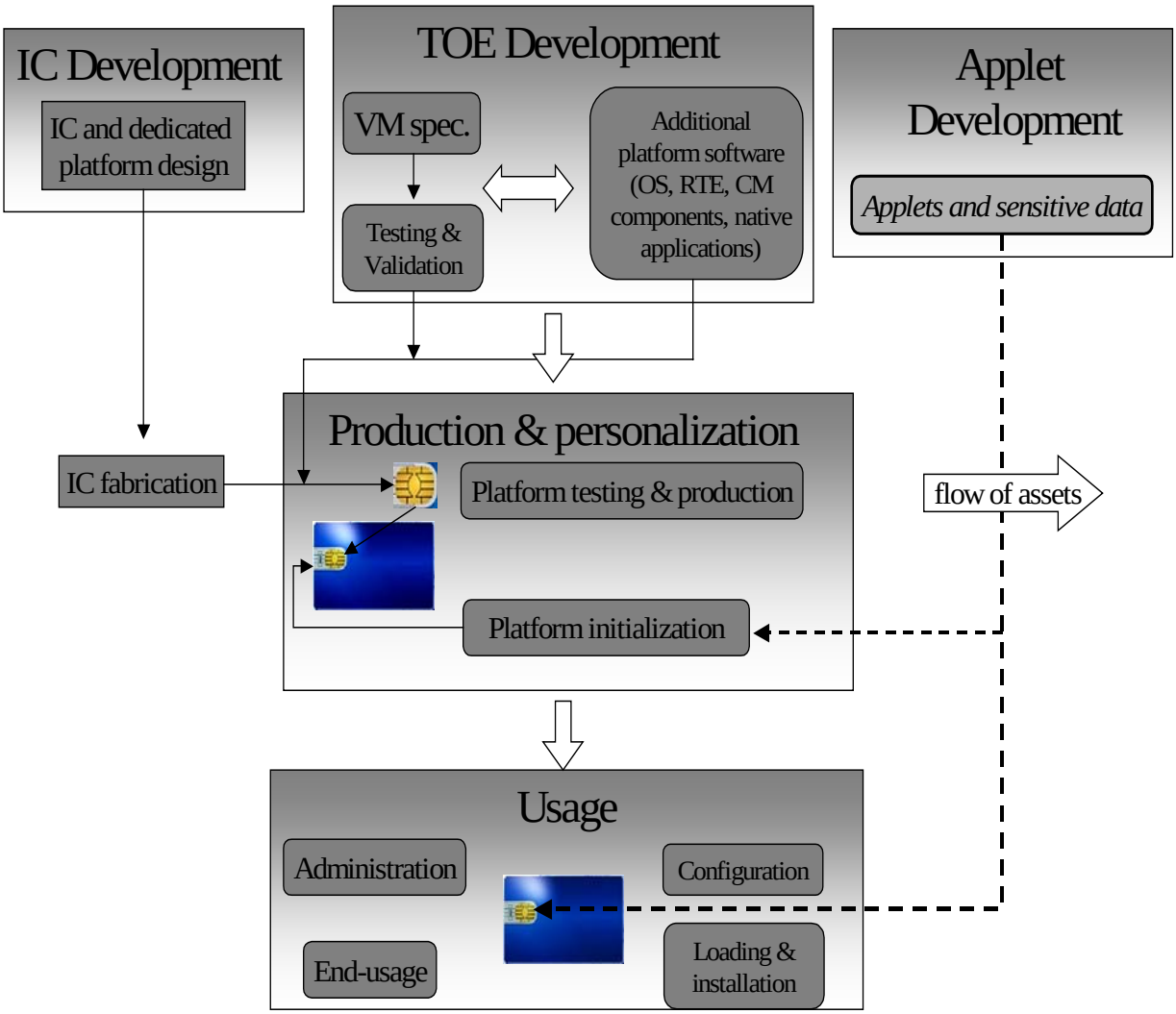

Figure 7: Smart Card Product Life Cycle

2.4.2.1 TOE Development & Production Environments

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

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

Development of the TOE then follows. The engineers use a secure computer system (preventing unauthorized access) to make their specifications, design, development and generation of the product. Storage of sensitive documents, databases on tapes, diskettes are in appropriately locked cupboards/safe. The disposal of unwanted data (complete electronic erasures) and documents (like shredding) is also of great importance. Testing, integration and validation of TOE components then take place. This phase consists in the collection of all software modules and the execution/test of this software on an emulator or on a simulator of the (DS & IC) layer.

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

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

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

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

2.4.2.2 TOE Final Environment

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

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

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

However, we can define the IT environment during this phase: first, the TOE obviously runs on top of what we called the *SCP*, and is itself part of the underlying platform for the card manager[6]. The underlying smart card platform has been described in §2.1.4 above. The TOE takes advantage of the features it provides for its own management needs, such as transaction facilities, memory management and safe cryptographic operations. At a lower level, the hardware provides physical protection of the TOE.

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

During normal usage, the card is inserted in a CAD or placed near a PCD, starting up the CM and *Java Card RE*. The session is an exchange of APDU commands between the CAD/PCD and the CM, the CM and the *Java Card RE* and, ultimately, the *Java Card RE* and some *applet*.

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

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

2.5 TOE INTENDED USAGE

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

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

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

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

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

So far, the most important applications are:

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

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

Java Card System 2.2 also provides *applet* deletion, which enables the fine tuning of open card management. This typically impacts the loyalty applications, which are obvious candidates for post-issuance downloading and removal of applications. Version 2.2.1 further restricts applet deletion rules and introduces a new optional method "uninstall" that is invoked by the Java Card RE before deleting the applet, provided the method is actually implemented.

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

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

2.6 PRODUCT RATIONALE

While the Java Card virtual machine (*Java Card VM*) is responsible for ensuring language-level security, the *Java Card RE* provides additional security features for Java Card technology-enabled devices.

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

The *applet firewall* is considered ~~as~~ the most important security feature. It enables complete isolation between *applet*s or controlled communication through additional mechanisms that allow them to share objects when needed. The *Java Card RE* allows such sharing using the concept of "shareable interface objects" (*SIO*) and **`static public`** variables. The *Java Card VM* should ensure that the only way for *applet*s to access any resources are either through the *Java Card RE* or through the Java Card platform API (or other vendor-specific APIs). This objective can only be guaranteed if *applet*s are correctly typed (all the "must clauses" imposed in chapter 7 of [JCVM21] on the bytecodes and the correctness of the *CAP* file format are satisfied).

3 TOE Security Environment

This chapter describes the security aspects of the environment in which the TOE is used. The first section describes some general security, and is intended to ease the comprehension of the security objectives and requirements, especially the access control policies. Sections §3.2 and §3.3 introduce the assets to be protected, the users of the TOE, and their software counterparts. Section §3.4 describes the assumptions made on the environment. Section §3.5 describes the threats menacing the assets of the TOE. Finally, the organizational policies that shall be imposed on the environment of the TOE are presented in Section §3.6.

All the sections in this chapter contain specific sub-sections for each of the TOE configurations introduced in Section §2.3.1.

3.1 SECURITY ASPECTS

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

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

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

This section presents several security aspects that will appear below in the configurations of the Protection Profile. Some being quite general, we give further details, which are numbered for easier cross-reference within the document. For instance, the two parts of #.OPERATE, when instantiated with an objective "O.OPERATE", may be met by separate SFRs in the rationale. The numbering then adds further details on the relationship between the objective and those SFRs.

CONFIDENTIALITY

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

#.CONFID-JCS-CODE *Java Card System* code must be protected against unauthorized disclosure. 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 API classes as well.

INTEGRITY

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

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

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

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

UNAUTHORIZED EXECUTIONS

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

#.EXE-JCS-CODE *Java Card System* (byte)code must be protected against unauthorized execution. *Java Card System* (byte)code includes any code of the *Java Card RE* or *API*. This concerns **(1)** invoking a method outside the scope of the

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

#.*FIREWALL* The *Java Card System* shall ensure controlled sharing of class instances[7], and isolation of their data and code between *package*s (that is, controlled execution contexts). **(1)** An *applet* shall neither read, write nor compare a piece of data belonging to an *applet* that is not in the same context, nor execute one of the methods of an applet in another context without its authorization.

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

BYTECODE VERIFICATION

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

CAP File Verification

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

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

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

the file in the card or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time.

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

Integrity and Authentication

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

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.

Linking and Verification

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

CARD MANAGEMENT

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

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

#.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). A change of identity, especially standing for an administrative role (like an *applet* impersonating the *Java Card RE*), is a severe violation of the TOE Security Policy (TSP). Selection controls the access to any data exchange between the TOE and the *CAD* and therefore, must be protected as well. The loading of a *package* or any exchange of data through the

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

#OBJ-DELETION **(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.

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

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

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

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.

#.*BIO-MNGT* The TOE shall provide a means to securely manage biometric templates. This includes: **(1)** Atomic update of biometric reference templates and try counter, **(2)** No rollback on the biometric-checking function,**(3)** Keeping the reference template (once initialized) secret (for instance, no clear-biometric-reading function), **(4)** Enhanced protection of biometric template's security attributes (state, try counter…) in confidentiality and integrity. This concerns version 2.2.2 of the Java Card platform.

#.*SCP* The smart card platform must be secure with respect to the TSP. 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 TSFs to be bypassed or altered and does not allow access to other low-level functions than those made available by the *package*s of the API. That includes the protection of its private data and code (against disclosure or modification) from the *Java Card System*. **(3)** It provides secure low-level cryptographic processing to the *Java Card System*. **(4)** It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism. **(5)** It allows the *Java Card System* to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low–level control accesses (segmentation fault detection). **(6)** It safely transmits low–level exceptions to the TOE (arithmetic exceptions, checksum errors), when applicable. We finally require that **(7)** the IC is designed in accordance with a well-defined set of policies and standards (likely specified in another protection profile), and will be tamper resistant to actually prevent an attacker from extracting or altering security data (like cryptographic keys) by using commonly employed techniques (physical probing and

sophisticated analysis of the chip). This especially matters to the management (storage and operation) of cryptographic keys.

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

3.2 ASSETS

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

Assets may overlap, in the sense that distinct assets may refer (partially or wholly) to the same piece of information or data. For example, "a piece of software" may be either 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.

3.2.1 User data

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

To be protected from unauthorized modification.

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

To be protected from unauthorized disclosure.

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

To be protected from unauthorized modification.

D.PIN Any end-user's PIN.

To be protected from unauthorized disclosure and modification.

D.BIO Any biometric template (this concerns version 2.2.2 of Java Card platform).

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

3.2.2 TSF data

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

To be protected from unauthorized disclosure and modification.

D.JCS_DATA The internal runtime data areas necessary for the execution of the *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 monopolization and unauthorized disclosure or modification.

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

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

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

To be protected from unauthorized disclosure and modification.

3.3 USERS & SUBJECTS

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

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

> *Package*s used on the Java Card platform that act on behalf of the applet developer. These subjects are involved in the ***FIREWALL security policy*** defined in §5.1.1.1 and they should be understood as instances of the subject `S.PACKAGE`.
>
> The *Java Card RE*, which acts on behalf of the card issuer. This subject is involved in several of the security policies defined in this document and is always represented by the subject `S.JCRE`.
>
> The bytecode verifier (*BCV*), which acts on behalf of the verification authority. This subject is involved in the ***PACKAGE LOADING security policy*** defined in §5.1.8 and is represented by the subject `S.BCV`.
>
> The *installer*, which acts on behalf of the card issuer. This subject is involved in the loading of *package*s and installation of *applet*s. It could play the role of the on-card entity in charge of package loading, which is involved in the ***PACKAGE LOADING security policy*** defined in §5.1.8 and is represented by the subject `S.CRD`.
>
> The *applet deletion manager*, if the configuration contains such components, which also acts on behalf of the card issuer. This subject is involved in the ***ADEL security policy*** defined in §5.1.4.1 and is represented by the subject `S.ADEL`.
>
> The *CAD* is involved in the ***Java Card RMI security policy*** defined in §5.1.5.1 and is represented by the subject `S.CAD`. It stands for both contacted and contacless devices.

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

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

3.4 ASSUMPTIONS

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

3.4.1 All Configurations

The following is an assumption for all the configurations:

A.NATIVE Those parts of the APIs written in native code as well as any pre-issuance native application on the card are assumed to be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p.41) for details.

3.4.2 Minimal Configuration

The assumptions of this configuration are the one defined in 3.4.1 plus the following ones:

A.NO-DELETION No deletion of installed *applet*s (or *package*s) is possible.

A.NO-INSTALL There is no post-issuance installation of *applet*s. Installation of *applet*s is secure and occurs only in a controlled environment in the pre-issuance phase. See #.*INSTALL* (p.42) for details.

A.VERIFICATION All the bytecodes are verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time.

3.4.3 Java Card System Standard 2.1.1 Configuration

The assumptions of the **Java Card System Standard 2.1.1** configuration are the one defined in 3.4.1 plus the following ones:

A.VERIFICATION As in the **Minimal** configuration.

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

A.DELETION Deletion of applets, if available through the card manager, is secure. Refer to #.*DELETION* for details on this assumption.

The rationale for this latter assumption is that even a Java Card System 2.1.1 TOE could be installed on a product that includes *applet* deletion features. This assumes that these functions are secure with respect to the TSPs herein.

3.4.4 Java Card System Standard 2.2 Configuration

The assumptions of this configuration are the one defined in §3.4.1 plus the following ones:

A.VERIFICATION As in the **Minimal** configuration.

A.APPLET As in the **Java Card System Standard 2.1.1** configuration.

3.4.5 Defensive Configuration

The assumption of this configuration is the one defined in 3.4.1.

3.5 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.

3.5.1 All Configurations

The following threats concern all the configurations considered in this document.

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

This threatens *all* the identified assets.

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

CONFIDENTIALITY

T.CONFID-JCS-CODE The attacker executes an application without authorization to disclose the *Java Card System* code. See *#.CONFID-JCS-CODE* (p. 39) for details.

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

T.CONFID-APPLI-DATA The attacker executes an application without authorization to disclose data belonging to another application. See #.*CONFID-APPLI-DATA* (p. 39) for details.

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

T.CONFID-JCS-DATA The attacker executes an application without authorization to disclose data belonging to the *Java Card System*. See #.*CONFID-JCS-DATA* (p. 40) for details.

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

INTEGRITY

T.INTEG-APPLI-CODE The attacker executes an application to alter (part of) its own or another application's code. See #.*INTEG-APPLI-CODE* (p. 40) for details.

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

T.INTEG-JCS-CODE The attacker executes an application to alter (part of) the *Java Card System* code. See #.*INTEG-JCS-CODE* (p. 40) for details.

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

T.INTEG-APPLI-DATA The attacker executes an application to alter (part of) another application's data. See #.*INTEG-APPLI-DATA* (p. 40) for details.

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

T.INTEG-JCS-DATA The attacker executes an application to alter (part of) *Java Card System* or API data. See #.*INTEG-JCS-DATA* (p. 40) for details.

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

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

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* (p. 42) 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 *Java Card RE* is usurped), **D.PIN**, **D.APP_KEYs** and **D.JCS_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* (p. 42) for further details. Directly threatened asset(s): **D.SEC_DATA** (any other asset may be jeopardized should this attack succeed, depending on whose identity was forged).

UNAUTHORIZED EXECUTION

T.EXE-CODE.1 An *applet* performs an unauthorized **execution** of a **method**. See #.*EXE-JCS-CODE* (p. 40) and #.*EXE-APPLI-CODE* (p. 40) 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* (p. 40) and #.*EXE-APPLI-CODE* (p. 40) for details.

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

T.NATIVE An *applet* **executes** a **native method** to bypass a security function such as the *firewall*. See #.*NATIVE* (p. 41) for details.

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

DENIAL OF SERVICE

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

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

3.5.2 Minimal Configuration

The threats of this configuration are the ones defined in §3.5.1.

3.5.3 Java Card System Standard 2.1.1 Configuration

The threats of this configuration are those defined in §3.5.1 plus the following ones:

INTEGRITY

T.INTEG-APPLI-CODE.2 The attacker modifies (part of) its own or another application code when an application *package* is transmitted to the card for installation. See #.*INTEG-APPLI-CODE* (p. 40) for details.

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

T.INTEG-APPLI-DATA.2 The attacker modifies (part of) the initialization data contained in an application *package* when the package is transmitted to the card for installation. See #.*INTEG-APPLI-DATA* (p. 40) for details.

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

MODIFICATIONS OF THE SET OF APPLICATIONS

T.INSTALL The attacker fraudulently **installs** post-issuance of an *applet* on the card. This concerns either the installation of an unverified *applet* or an attempt to induce a malfunction in the TOE through the installation process. See #.*INSTALL* (p 42) for details.

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

3.5.4 Java Card System Standard 2.2 Configuration

The threats of this configuration are those defined in §3.5.1 plus the following ones:

T.INTEG-APPLI-CODE.2 As in the **Java Card System Standard 2.1.1** configuration.

T.INTEG-APPLI-DATA.2 As in the **Java Card System Standard 2.1.1** configuration.

T.INSTALL As in the **Java Card System Standard 2.1.1** configuration.

UNAUTHORIZED EXECUTIONS

T.EXE-CODE-REMOTE The attacker performs an unauthorized **remote execution** of a **method** from the *CAD*. See #.*EXE-JCS-CODE* (p. 40) and #.*EXE-APPLI-CODE* (p. 40) for details.

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

This threat concerns version 2.2 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 TSC.

CARD MANAGEMENT

T.DELETION The attacker **deletes** an *applet* or a *package* already in use on the card, or uses the deletion functions to pave the way for further attacks (putting the TOE in an insecure state). See #.*DELETION* (p 43) for details).

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

T.OBJ-DELETION The attacker keeps a reference to a garbage collected object in order to force the TOE to execute an unavailable method, to make it to crash, or to gain access to a memory containing data that is now being used by another application. See #.*OBJ-DELETION* (p. 43) for further details.

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

3.5.5 Defensive Configuration

The threats of this configuration are those defined in §3.5.1 plus the following ones:

T.INSTALL As in the **Java Card System Standard 2.1.1** configuration.

T.EXE-CODE-REMOTE As in the **Java Card System Standard 2.2** configuration.

T.DELETION As in the **Java Card System Standard 2.2** configuration.

T.OBJ-DELETION As in the **Java Card System Standard 2.2** configuration.

3.6 ORGANIZATIONAL SECURITY POLICIES

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

3.6.1 Minimal Configuration

There is no organizational security policy for this configuration.

3.6.2 Java Card System Standard 2.1.1 Configuration

This configuration has only one organizational security policy:

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* (p.*41*) for details.

3.6.3 Java Card System Standard 2.2 Configuration

This configuration has only one organizational security policy:

OSP.VERIFICATION As in the **Java Card System Standard 2.1.1** configuration.

3.6.4 Defensive Configuration

There is no organizational security policy for this configuration.

4 SECURITY OBJECTIVES

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

4.1 SECURITY OBJECTIVES FOR THE TOE

4.1.1 All Configurations

The following are security objectives of all the configurations considered in this document.

IDENTIFICATION

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

EXECUTION

O.OPERATE The TOE must ensure continued correct operation of its security functions. See #.*OPERATE* (p 43) for details.

O.RESOURCES The TOE shall control the availability of resources for the applications. See #.*RESOURCES* (p 43) for details.

O.FIREWALL The TOE shall ensure controlled sharing of data containers owned by *applets* of different *packages*, and between *applets* and the TSFs. See #.*FIREWALL* (p 41) for details.

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* (p 41) 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.

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

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

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

SERVICES

O.ALARM The TOE shall provide appropriate feedback information upon detection of a potential security violation. See #.*ALARM* (p. 43) for details.

O.TRANSACTION The TOE must provide a means to execute a set of operations atomically. See #.*TRANSACTION* (p. 45) for details.

O.CIPHER The TOE shall provide a means to cipher sensitive data for applications in a secure way. In particular, the TOE must support cryptographic algorithms consistent with cryptographic usage policies and standards. See #.*CIPHER* (p. 44) for details.

O.PIN-MNGT The TOE shall provide a means to securely manage PIN objects. See #.*PIN-MNGT* (p. 44) for details.

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

O.KEY-MNGT The TOE shall provide a means to securely manage cryptographic keys. This concerns the correct generation, distribution, access and destruction of cryptographic keys. See #.*KEY-MNGT* (p. 44).

Application note: *O.KEY-MNGT*, *O.PIN-MNGT*, *O.TRANSACTION* and *O.CIPHER* are actually provided to *applet*s in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to *applet*s; those may be built on top of the Java Card API or independently. Depending on whether they contain native code or not, these proprietary libraries will need to be evaluated together with the TOE or not

(see #.*NATIVE*, p.41). In any case, they are not included in the *Java Card System* for the purpose of the present document.

4.1.2 Minimal Configuration

The security objectives of this configuration are the ones defined in §4.1.1.

4.1.3 Java Card System Standard 2.1.1 Configuration

The security objectives of this configuration are the ones defined in §4.1.1 plus the following ones:

APPLET MANAGEMENT	
O.INSTALL	The TOE shall ensure that the installation of an *applet* performs as expected. See #.*INSTALL* (p 42 for details).
O.LOAD	The TOE shall ensure that the loading of a *package* into the card is safe.

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

4.1.4 Java Card System Standard 2.2 Configuration

The security objectives of this configuration are the ones defined in §4.1.1 plus the following ones:

O.INSTALL	As in the **Java Card System Standard 2.1.1** configuration
O.LOAD	As in the **Java Card System Standard 2.1.1** configuration

APPLET MANAGEMENT	
O.DELETION	The TOE shall ensure that both *applet* and *package* deletion perform as expected. See #.*DELETION* (p 43) for details.

OBJECT DELETION	
O.OBJ-DELETION	The TOE shall ensure the object deletion shall not break references to objects. See #.*OBJ-DELETION* (p. 43) for further details.

SERVICES

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 of the Java Card platform.

O.BIO-MNGT The TOE shall provide a means to securely manage biometric templates. This concerns the optional package javacardx.biometry of version 2.2.2 of the Java Card platform. See #.*BIO-MNGT* for details.

O.EXT_MEM The TOE shall provide controlled access means to the external memory and ensure that the external memory does not address Java Card System memory (containing User Data and TSF Data). This concerns the optional package javacardx.external of version 2.2.2 of the Java Card platform.

4.1.5 Defensive Configuration

The security objectives of this configuration are the ones defined in §4.1.1 plus the following ones:

O.INSTALL As in the **Java Card System Standard 2.1.1** Configuration.

O.1186HDELETION As in the **Java Card System Standard 2.2** configuration.

O.OBJ-DELETION As in the **Java Card System Standard 2.2** configuration.

O.REMOTE As in the **Java Card System Standard 2.2** configuration.

O.BIO-MNGT As in the **Java Card System Standard 2.2** configuration.*O.EXT_MEM* As in the **Java Card System Standard 2.2** configuration.

INTEGRITY, CONFIDENTIALITY AND CORRECT EXECUTION

O.VERIFICATION The TOE shall ensure that any bytecode is verified prior to being executed. See #.*VERIFICATION* (p.*41*) for details.

4.2 SECURITY OBJECTIVES FOR THE ENVIRONMENT

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

4.2.1 All Configurations

The following objectives are common to all the configurations considered in this document.

OE.NATIVE Those parts of the APIs written in native code as well as any pre-issuance native application on the card shall be conformant with the TOE so as to ensure that security policies and objectives described herein are not violated. See #.*NATIVE* (p.41) for details.

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

OE.SCP.SUPPORT The *SCP* shall provide functionalities that support the well-functioning of the TSFs of the TOE (avoiding they are bypassed or altered) and by controlling the access to information proper of the TSFs. In addition, the smart card platform should also provide basic services which are required by the runtime environment to implement security mechanisms such as atomic transactions, management of persistent and transient objects and cryptographic functions. These mechanisms are likely to be used by security functions implementing the security requirements defined for the TOE. See #.*SCP.2-5* (p.44).

OE.SCP.IC The *SCP* shall possess IC security features. See #.*SCP.7* (p.44).

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

As already mentioned in §2.1.3 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 (*applet*s). The card manager should prevent card content management (loading, installation, deletion) at invalid states of the card or carried out by non-authorized actors. It shall also enforce security policies established by the card issuer.

These environmental objectives shall be met by IT security requirements.

4.2.2 Minimal Configuration

The objectives for the environment in this configuration are those defined in §4.2.1 plus the following ones:

OE.NO-DELETION No installed *applet*s (or *package*s) shall be deleted from the card.

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

The objectives *OE.NO-INSTALL* and *OE.NO-DELETION* have been included so as to describe procedures that shall contribute to ensure that the TOE will be used in a secure manner. Moreover, they have been defined in accordance with the environmental assumptions they uphold (actually, they are just a reformulation of the corresponding assumptions). The ***NO-DELETION*** and ***NO-INSTALL*** (assumptions and objectives) constitute the explicit statement that the **Minimal**

configuration corresponds to that of a closed card (no code can be loaded or deleted once the card has been issued). It is not evident that IT means should be used to carry out these objectives.

OE.VERIFICATION All the bytecodes shall be verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time. See #.*VERIFICATION* (p.41) for details.

4.2.3 Java Card System Standard 2.1.1 Configuration

The objectives for the environment in this configuration are those defined in §4.2.1 plus the following ones:

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

OE.VERIFICATION As in the **Minimal** Configuration.

4.2.4 Java Card System Standard 2.2 Configuration

The objectives for the environment in this configuration are those defined in §4.2.1 plus the following ones:

OE.APPLET As in the **Java Card System Standard 2.1.1** Configuration.

OE.VERIFICATION As in the **Minimal** Configuration.

4.2.5 Defensive Configuration

The objectives for the environment in this configuration are those defined in §4.2.1.

5 IT SECURITY REQUIREMENTS

This section defines the detailed security requirements that shall be satisfied by each configuration of the TOE and its respective IT environment. As explained in Section §1.7.1, they are arranged into several groups, which are then composed to form the different configurations of the TOE. Depending on the configuration, the groups of SFRs are either TOE SFRs or SFRs on the IT environment (see Table 2 below).

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

5.1 TOE AND IT ENVIRONMENT SECURITY REQUIREMENTS

The following table (already presented and described in §2.4.1.1) displays the relationship between the chosen configurations and the groups that shall be defined in the sections that follow.

Group (group name)	Minimal	Java Card System Standard 2.1.1	Java Card System Standard 2.2	Defensive
Core (*CoreG*)	TOE	TOE	TOE	TOE
Smart card platform (*SCPG*)	IT	IT	IT	IT
Installer (*InstG*)	--	TOE	TOE	TOE
RMI (*RMIG*)	--	--	TOE	TOE
Logical channels (*LCG*)	--	--	TOE	TOE
Object deletion (*ODELG*)	--	--	TOE	TOE
Bytecode verification (*BCVG*)	IT	IT	IT	TOE
Applet deletion (*ADELG*)	--	--	TOE	TOE
Secure carrier (*CarG*)	--	TOE	TOE	--
Card manager (*CMGRG*)	IT	IT	IT	IT
External Memory (*EMG*)	--	--	TOE	TOE

Table 2: Relationship between Groups and Configurations

5.1.1 *CoreG* Security Functional Requirements

This group is focused on the main security policy of the Java Card System, known as the *firewall*. This policy essentially concerns the security of *installed applet*s. The policy focuses on the execution of bytecodes.

5.1.1.1 Firewall Policy

FDP_ACC.2: COMPLETE ACCESS CONTROL

FDP_ACC.2.1 **The TSF shall enforce the [assignment: *access control SFP*] on [assignment: *list of subjects and objects*] and all operations among subjects and objects covered by the SFP.**

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

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

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

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

Operation	Description
`OP.ARRAY_ACCESS(O.JAVAOBJECT,` field`)`	Read/Write an array component.
`OP.INSTANCE_FIELD(O.JAVAOBJECT,` field`)`	Read/Write a field of an instance of a class in the Java programming language

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

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

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

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

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

FDP_ACF.1.1 **The TSF shall enforce the [assignment: *access control SFP*] to objects based on the following: [assignment: *list of subjects and objects controlled under the indicated SFP, and for each, the SFP-relevant security attributes, or named groups of SFP-relevant security attributes*].**

FDP_ACF.1.1/*FIREWALL* The TSF shall enforce the ***FIREWALL access control SFP*** to objects based on the following: ***subjects, objects and their security attributes, described hereafter.***

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

Subject/Object	Attributes
`S.PACKAGE`	Context, Selected, Active
`S.JCRE`	Active

Subject/Object	Attributes
O.JAVAOBJECT	Sharing, Context, LifeTime

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

Name	Description
Context	*Package AID*, or "Java Card RE"
Sharing	Standard, SIO, Java Card RE entry point, or global array
LifeTime	**CLEAR_ON_DESELECT** or **PERSISTENT**.[8]
Selected, Active	*Boolean value: true or false*

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

The Sharing attribute defines four categories of objects:

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

When a new object is created, it is associated with the currently active context. But the object is owned by the *applet* instance within the currently active context when the object is instantiated ([JCRE21], §6.1.2). An object is owned by an *applet* instance, by the *Java Card RE* or by the *package* library where it has been defined (these latter objects can only be arrays that initialize static fields of packages).

Finally both "the currently active context" and "the SELECTed applet context" are security attributes internal to the Java Card VM.

> ([JCRE21], Glossary) *Currently selected applet*. 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 currently selected applet. The Java Card RE sends all APDU commands to the currently selected applet.

[8] *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.

While the expression "selected applet" 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 package whose *Selected* attribute holds *true*.

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

In this policy, the "active context" is the package or Java Card RE whose *Active* security attribute holds *true*.

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

FDP_ACF.1.2 **The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [assignment: *rules governing access among controlled subjects and controlled objects using controlled operations on controlled objects*].**

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

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

R.JAVA.2 ([JCRE21]§6.2.8) The currently active *`S.PACKAGE`* may freely perform any of *`OP.ARRAY_ACCESS`*, *`OP.INSTANCE_FIELD`*, *`OP.TYPE_ACCESS`*, *`OP.INVK_VIRTUAL`*, *`OP.INVK_INTERFACE`* or *`OP.THROW`* upon any *`O.JAVAOBJECT`* whose Sharing attribute has value "**Standard**" and whose Lifetime attribute has value "**PERSISTENT**" only if *`O.JAVAOBJECT`*'s Context attribute has the same value as the active context.

R.JAVA.3 ([JCRE21]§6.2.8.10) The currently active *`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 ([JCRE21]§6.2.8.6) The currently active *`S.PACKAGE`* may perform *`OP.INVK_INTERFACE`* upon an *`O.JAVAOBJECT`* whose Sharing attribute has the value "**SIO**" only if the invoked *interface* method extends the **`Shareable`** *interface*.

R.JAVA.5 The currently active *S.PACKAGE* may perform an *OP.CREATE* only if the value of the Sharing parameter[9] is "Standard".

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

FDP_ACF.1.3 **The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [assignment:** ***rules, based on security attributes, that explicitly authorize access of subjects to objects*****].**

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

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

FDP_ACF.1.4 **The TSF shall explicitly deny access of subjects to objects based on the [assignment:** ***rules, based on security attributes, that explicitly deny access of subjects to objects*****].**

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

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

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

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

FDP_IFC.1 SUBSET INFORMATION FLOW CONTROL

FDP_IFC.1.1 **The TSF shall enforce the [assignment:** ***information flow control SFP*****] on [assignment:** ***list of subjects, information, and operations that cause controlled information to flow to and from controlled subjects covered by the SFP*****].**

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

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

Subjects[10] (prefixed with an "*S*") and information (prefixed with an "*I*") covered by this policy are:

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

There is a unique operation in this policy:

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

Application note: References of temporary *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(S_1,S_2,I)* operations as well.

FDP_IFF.1 SIMPLE SECURITY ATTRIBUTES

FDP_IFF.1.1 **The TSF shall enforce the [assignment:** ***information flow control SFP*****] based on the following types of subject and information security attributes: [assignment:** ***list of subjects and information controlled under the indicated SFP, and for each, the security attributes*****].**

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

FDP_IFF.1.2 **The TSF shall permit an information flow between a controlled subject and controlled information through a controlled operation if the following rules hold: [assignment:** ***for each operation, the security attribute-based relationship that must hold between subject and information security attributes*****].**

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

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

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

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

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

FDP_IFF.1.4 **The TSF shall provide the following [assignment:** *list of additional SFP capabilities*]**.**

FDP_IFF.1.4/JCVM The TSF shall provide the following ***[assignment: list of additional SFP capabilities]***.

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

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

FDP_IFF.1.6 **The TSF shall explicitly deny an information flow based on the following rules: [assignment:** *rules, based on security attributes, that explicitly deny information flows*]**.**

FDP_IFF.1.6/JCVM The TSF shall explicitly deny an information flow based on the following rules: ***[assignment: other 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 ([JCRE21], §6.2.8.1-3).

Note 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.1.3/1267HJCVM*** to ***FDP_IFF.1.6/1270HJCVM*** 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 trough an internal register prior to being pushed on the stack of the invoker. The **areturn** bytecode would cause more than one *OP.PUT* operation under this scheme.

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

FDP_RIP.1.1 **The TSF shall ensure that any previous information content of a resource is made unavailable upon the [selection: *allocation of the resource to, de-allocation of the resource from*] the following objects: [assignment: *list of objects*].**

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

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

FMT_MSA.1.1 **The TSF shall enforce the [assignment: access control SFP, information flow control SFP] to restrict the ability to [selection: *change default, query, modify, delete,* [assignment: *other operations*]] the security attributes [assignment: *list of security attributes*] to [assignment: *the authorized identified roles*].**

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

Application note: The modification of the active context as well as that of the selected applet should be performed in accordance with the rules given in [JCRE21], §4 and [JCVM21], §3.4.

FMT_MSA.2 SECURE SECURITY ATTRIBUTES

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

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

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 a `*.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 STATIC ATTRIBUTE INITIALIZATION

FMT_MSA.3.1 **The TSF shall enforce the [assignment: *access control SFP, information flow control SFP*] to provide [selection: *restrictive, permissive, [assignment:other property]*] default values for security attributes that are used to enforce the *SFP*.**

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

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

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

FMT_MSA.3.2 **The TSF shall allow the [assignment: *the authorized identified roles*] to specify alternative initial values to override the default values when an object or information is created.**

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

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

FMT_SMR.1 SECURITY ROLES

FMT_SMR.1.1 **The TSF shall maintain the roles: [assignment: *the authorized identified roles*].**

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

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

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

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1 **The TSF shall be capable of performing the following security management functions: [assignment: list of security management functions to be provided by the TSF].**

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

- ***Modify the active context and the SELECTed applet Context.***
- ***Modify the list of registered applets' AID.***

FPT_SEP.1 TSF DOMAIN SEPARATION

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

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

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

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

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

5.1.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

FCS_CKM.1 CRYPTOGRAPHIC KEY GENERATION

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

FCS_CKM.1.1 The TSF shall generate cryptographic KEYS in accordance with a specified cryptographic KEY generation algorithm [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 [JCAPI21] specification in *class*es **KeyBuilder** and **KeyPair** (at least Session key generation).

Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI21][JCAPI22], [JCAPI221], [JCAPI222]).

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 [JCAPI21] standard.

Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI21], [JCAPI22], [JCAPI221], [JCAPI222]).

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 in accordance with [JCAPI21] in *class* **Key**.

Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI21], [JCAPI22], [JCAPI221], [JCAPI222]).

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

Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI21], [JCAPI22], [JCAPI221], [JCAPI222]).

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 [JCAPI21] in accordance to [JCAPI21] specification (see **javacardx.crypto.Cipher** and **javacardx.security** packages).

Application note: This component shall be instantiated according to the version of the Java Card API applying to the security target and the implemented algorithms ([JCAPI21], [JCAPI22], [JCAPI221], [JCAPI222]).

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

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

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

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

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

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

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

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

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

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

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

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

FDP_ROL.1 BASIC ROLLBACK

FDP_ROL.1.1 **The TSF shall enforce [assignment:** *access control SFP(s) and/or information flow control SFP(s)***] to permit the rollback of the [assignment:** *list of operation*s**] on the [assignment: information and/or** *list of object*s**].**

FDP_ROL.1.2 **The TSF shall permit operations to be rolled back within the [assignment:** *boundary limit to which rollback may be performe*d**].**

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

Application note: ***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*** [JCRE21], §7.7, within the bounds of ***the Commit Capacity*** ([JCRE21], §7.8), ***and those described in*** [JCAPI21], [JCVM221]Transactions are a service offered by the APIs to *applet*s. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [JCAPI21] (see for instance, PIN-blocking, PIN-checking, update of *Transient object*s).

Application note: The rollback within the scope of the uninstall() method only applies to Java Card platform, version 2.2.1 compliant TOEs.

5.1.1.3 Card Security Management

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

FAU_ARP.1 SECURITY ALARMS

FAU_ARP.1.1 **The TSF shall take [assignment: *list of the least disruptive actions*] upon detection of a potential security violation.**

FAU_ARP.1.1/JCS The TSF shall ***throw an exception, lock the card session or reinitialize the Java Card System and its data [assignment: other actions]*** upon detection of a potential security violation.

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

- *CAP file* inconsistency
- Typing error in the operands of a bytecode
- *applet* life cycle inconsistency
- Card tearing (unexpected removal of the Card out of the CAD or RF signal loss) and power failure
- Abortion of a transaction in an unexpected context (see (**`abortTransaction()`**, [JCAPI21] and ([JCRE21], §7.6.2)
- Violation of the Firewall or Java Card VM SFPs
- Unavailability of resources
- Array overflow
- Other runtime errors related to *applet*'s failure (like uncaught exceptions)

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

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

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

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

The locking may be implemented at the level of the *Java Card System* as a denial of service (through some systematic "fatal error" message or return value) that lasts up to the next "RESET" event, without affecting other components of the card (such as the card manager).

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

FDP_SDI.2 STORED DATA INTEGRITY MONITORING AND ACTION

FDP_SDI.2.1 The TSF shall monitor user data stored within the TSC 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 specification, at least an exception shall be raised upon integrity errors detection on cryptographic keys, PIN values, biometric templatesand their associated security attributes. Even if all the objects cannot be monitored, cryptographic keys, PIN objects and biometric shall be considered with particular attention by ST authors as they play a key role in the overall security.

Application note: 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.

FPT_RVM.1 NON-BYPASSABILITY OF THE TSP

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

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

FPT_TDC.1 INTER-TSF BASIC TSF DATA CONSISTENCY

FPT_TDC.1.1 **The TSF shall provide the capability to consistently interpret [assignment: list of TSF data types] when shared between the TSF and another trusted IT product.**

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

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

FPT_TDC.1.2 **The TSF shall use [assignment: *list of interpretation rules to be applied by the TSF*] when interpreting the TSF data from another trusted IT product.**

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

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

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

FPT_FLS.1.1 **The TSF shall preserve a secure state when the following types of failures occur: [assignment: *list of types of failures in the TSF*].**

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

Application note: The *Java Card RE Context* is the *Current context* when the Java Card VM begins running after a card reset ([JCRE21], §6.2.3) or after a proximity card (PICC) activation sequence ([JCRE222]). Behavior of the TOE on power loss and reset is described in [JCRE21], §3.5, and §7.1. Behavior of the TOE on RF signal loss is described in [JCRE222], §3.6.2.

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 [JCRE21] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.

FPT_TST.1 TSF TESTING

FPT_TST.1.1 **The TSF shall run a suite of self-tests [selection: during initial start-up, periodically during normal operation, at the request of the authorized user, at the conditions [assignment: conditions under which self test should occur]] to demonstrate the correct operation of the TSF.**

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

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

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

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

5.1.1.4 AID Management

FMT_MTD.1 MANAGEMENT OF TSF DATA

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

FMT_MTD.1.1 **The TSF shall restrict the ability to [selection: change default, query, modify, delete, clear, [assignment: other operations]] the [assignment: list of TSF data] to [assignment: the authorized identified roles].**

FMT_MTD.1.1/JCRE The TSF shall restrict the ability to **modify** the **list of registered applets'** *AID* to **the Java Card RE** ***[assignment:*** *other authorized identified role****]***.

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

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

FMT_MTD.3 SECURE TSF DATA

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

FIA_ATD.1 User attribute definition

FIA_ATD.1.1 **The TSF shall maintain the following list of security attributes belonging to individual users: [assignment: *list of security attributes*].**

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

FIA_UID.2 User identification before any action

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

Application note: By users here it must be understood the ones associated to the *packages (or applets)* 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.

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

FIA_USB.1 User-subject binding

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

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

5.1.2 *InstG* Security Functional Requirements

This group bulks the SFRs related to the installation of the *applet*s, which addresses security aspects outside the runtime. The idea here is that installation of *applet*s is a critical phase, which lies partially out of the boundaries of the *firewall*, and therefore has to be deserved specific treatment. In the Common Criteria model, loading a *package* or installing an *applet* was considered as being an importation of user data (that is, user application's data) with its security attributes (such as the parameters of the *applet* used in the firewall rules).

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

FDP_ITC.2 IMPORT OF USER DATA WITH SECURITY ATTRIBUTES

FDP_ITC.2.1 **The TSF shall enforce the [assignment: *access control SFP(s) and/or information flow control SFP(s)]* when importing user data, controlled under the SFP, from outside of the TSC.**

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

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

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

FDP_ITC.2.3/Installer The TSF shall ensure that the protocol used provides for the unambiguous association between the security attributes and the user data received.

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

FDP_ITC.2.4/Installer The TSF shall ensure that interpretation of the security attributes of the imported user data is as intended by the source of the user data.

Application note: Each package contains a package Version attribute, which is a pair of major and minor version numbers ([JCVM21], §4.5). With the AID, it describes the package defined in the CAP file. When an export file is used during preparation of a CAP file, the versions numbers and AIDs indicated in the export file are recorded in the CAP files ([JCVM21], §4.5.2): the dependent packages Versions and AIDs attributes allow the retrieval of these identifications.. Implementation-dependent checks may occur on a case-by-case basis to indicate that *package* files are binary compatibles. However, *package* files do have "*package* Version Numbers" ([JCVM21])

used to indicate binary compatibility or incompatibility between successive implementations of a *package*, which obviously directly concern this requirement.

FDP_ITC.2.5 **The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TSC: [assignment: *additional importation control rules*].**

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

A *package* may depend on (import or use data from) other *package*s already installed. This dependency is explicitly stated in the loaded *package* in the form of a list of *package AIDs*. The loading is allowed only if, for each dependent *package*, its *AID* attribute is equal to a resident package *AID* attribute, the major (minor) Version attribute associated to the former is equal (less than or equal) to the major (minor) Version attribute associated to the latter ([JCVM21],§4.5.2).

Application note: The intent of this rule is to ensure the binary compatibility of the *package* with those already on the card ([JCVM21], §4.4).

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

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

FMT_SMR.1 Security roles

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

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

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

FPT_FLS.1 Failure with preservation of secure state

FPT_FLS.1.1/Installer The TSF shall preserve a secure state when the following types of failures occur: *the installer fails to load/install a package/applet as described in [JCRE21] §10.1.4.*

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

FPT_RCV.3 AUTOMATED RECOVERY WITHOUT UNDUE LOSS

FPT_RCV.3.1/Installer When automated recovery from [assignment: *list of failures/service discontinuities*] is not possible, the TSF shall enter a maintenance mode where the ability to return the TOE to a secure state is provided.

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

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

FPT_RCV.3.2/Installer For ***[assignment: list of failures/service discontinuities],*** the TSF shall ensure the return of the TOE to a secure state using automated procedures.

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

Application note: In the case where the configuration includes the *applet deletion manager* (and the associated group, ADELG), this component shall include among the listed failures that of the deletion of a *package*/*applet*. See ([JCRE22], 11.3.4) for possible scenarios. Precise behavior is left to implementers.

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

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

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

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

FRU_RSA.1 MAXIMUM QUOTAS

FRU_RSA.1.1 **The TSF shall enforce maximum quotas of the following resources: [assignment: *controlled resources*] that [selection: *individual user, defined group of users, subjects*] can use [selection: *simultaneously, over a specified period of time*].**

FRU_RSA.1.1/Installer The TSF shall enforce maximum quotas of the following resources: **persistent memory** *[other controlled resources]* that **applet instances** can use **simultaneously.**

Application note: TOEs that are compliant with Global Platform specifications should implement these quota checks at installation time.

5.1.3 *BCVG* Security Functional Requirements

This group of requirements concerns bytecode verification. A bytecode verifier can be understood as a process that acts as a filter on a *CAP* file verifying that the bytecodes of the methods defined in the file conform to certain well-formed requirements. As mentioned in §2.1.1, there are different techniques that have been proposed for performing those checks. The solution described in [JCBV], for example, is based on a data flow analysis and makes use of an abstract interpreter. The abstract interpreter simulates execution of each instruction, using types of the data being operated on instead of values. For each instruction, the state of the operand stack and local variables are compared to the type(s) required during execution, and then are updated according to the operation of the instruction. The main component of this group of functional requirements is an information flow control policy, which describes the constraints imposed on the operations (the bytecodes) that make information flow between the subjects (local variables, operand stack, fields).

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

FDP_IFC.2 COMPLETE INFORMATION FLOW CONTROL

FDP_IFC.2.1 **The TSF shall enforce the [assignment: *information flow control SFP*] on [assignment: *list of subjects* and *information*] and all operations that cause that information to flow to and from subjects covered by the SFP.**

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

Subjects[11] (prefixed with an "*S*") covered by this policy are:

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

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

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

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

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

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

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

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

FDP_IFF.2 HIERARCHICAL SECURITY ATTRIBUTES

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

FDP_IFF.2.1 **The TSF shall enforce the [assignment: *information flow control SFP*] based on the following types of subject and information security attributes: [assignment: *the minimum number and type of security attributes*].**

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

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

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

The following table describes the security attributes.

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

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

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

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

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

Byte values have type ***byte*** and short values have type ***short***. The first and second halves of an integer value has respectively type ***int***$_1$, and ***int***$_2$. The type of a reference to an instance of the class *C* is ***C*** itself. A reference to an array of elements of type *T* has type ***T[]***. From the previous basic types it is possible to build the type T_0 (T_1 x_1, …. T_n x_n) of a method. A return address *adrs* of a subroutine has type ***RetAddrss(****adrs****)***. Finally, the former types are extended with three extra types **Top**, **Null** and , so that the domain of types forms a complete lattice. **Top** is the type of any piece of data, that is, the maximum of the lattice. **Null** is the type of the default value null of all the reference types (classes and arrays). is the type of an element that belongs to all types (for instance the value 0, provided that null is represented as zero).

FDP_IFF.2.2 **The TSF shall permit an information flow between a controlled subject and controlled information through a controlled operation if the following rules, based on the ordering relationships between security attributes hold: [assignment: for each operation, the security attribute-based relationship that must hold between subject and information security attributes].**

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

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

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

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

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

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

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

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

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

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

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

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

FDP_IFF.2.3 **The TSF shall enforce the [assignment:** ***additional information flow control SFP rules*****].**

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

FDP_IFF.2.4 **The TSF shall provide the following [assignment:** ***list of additional SFP capabilities*****].**

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

FDP_IFF.2.5 **The TSF shall explicitly authorize an information flow based on the following rules: [assignment:** ***rules, based on security attributes that explicitly authorize information flows*****].**

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

FDP_IFF.2.6 **The TSF shall explicitly deny an information flow based on the following rules: [assignment:** ***rules, based on security attributes that explicitly deny information flows*****].**

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

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

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

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

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

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

Top is the maximum of all types;

Null is the minimum of all classes and array types;

is the minimum of all types.

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

(See ***FMT_SMR.1.1/BCV*** (p. 91) for the roles)

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

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

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

FMT_MSA.2 SECURE SECURITY ATTRIBUTES

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

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

Correspondences between the different components of the CAP file (for instance, each class in the class component has an entry in the descriptor component).

Pointer soundness (example: the index argument in a static method invocation always has an entry in the constant pool);

Absence of hanged pointers (example: each exception handler points to the beginning of some bytecode);

Redundant information (enabling different ways of searching for it);

Conformance to the Java Language Specification respecting the access control features mentioned in §2.2 of [JCVM22].

Packages that are loaded post-issuance cannot contain native code.

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

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

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

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

FMT_SMR.1 Security roles

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

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

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

FMT_SMF.1 Specification of Management Functions

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

FRU_RSA.1 Maximum quotas

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

5.1.4 *ADELG* Security Functional Requirements

This group bulks the SFRs related to the deletion of *applet*s and/or *packages*, enforcing the *applet deletion manager* **(ADEL) policy** on security aspects outside the runtime. The idea here is that deletion is a critical phase and therefore requires specific treatment. This policy is better thought as a frame to be filled by ST implementers.

5.1.4.1 Applet Deletion Manager Policy

FDP_ACC.2: COMPLETE ACCESS CONTROL

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

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

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

In Java Card version 2.2.1, S.ADEL calls the "unistall" method of the applet instance to be deleted, if implemented by the applet, to inform it of the deletion request. The order in which these calls and the dependencies checks are performed are out of the scope of this protection profile: `O.CODE_PKG` The code of a *package*, including alllinking information. On the Java Card platform, a package is the installation unit.

`O.APPLET` Any installed *applet*, its code and data.

`O.JAVAOBJECT` Java class instance or array.

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

Operation	Description
`OP.DELETE_APPLET(O.APPLET,…)`	Delete an installed *applet* and its objects, either logically or physically.

Operation	Description
OP.DELETE_PCKG(O.CODE_PKG,...)	Delete a *package*, either logically or physically
OP.DELETE_PCKG_APPLET(O.CODE_PKG,...)	Delete a *package* and its installed *applet*s, either logically or physically.

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

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

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

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

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

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

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

An applet instance can be in two different selection states: selected or deselected. If the applet is selected (in some logical channel), then in turn it could either be *currently selected* or just *active*. At any time there could be more than one active applet instances over each I/O interface, but only one currently selected (the maximum number of active applet instances depends on the Java Card platform version). This latter is the one that is processing the current command ([JCRE22], §4).

The Owner of an object is either the applet instance that created the object or the package (library) where it has been defined (these latter objects can only be arrays that initialize static fields of the package).

An object is said to be remote if it is an instance of a class that directly or indirectly implements the interface **`java.rmi.Remote`**.

Finally, there are needed security attributes that are not attached to any object or subject of the TSP: (1) the ResidentPackages Versions (or Resident Image,[JCVM21],§4.5) and *AID*s. They describe the packages that are already on the card, (2) the list of registered applet instances and (3) the ActiveApplets security attribute. They are all attributes internal to the Java Card VM, that is, not attached to any specific object or subject of the SPM. These attributes are TSF data that play a role in the SPM.

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

The subject of this policy is `S.ADEL`.

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

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

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

R.JAVA.14 ([JCRE22], §11.3.4.1, **Applet Instance Deletion**). The `S.ADEL` may perform `OP.DELETE_APPLET` upon an `O.APPLET` only if, (1) `S.ADEL` is currently selected, (2) `O.APPLET` is deselected and (3) there is no `O.JAVAOBJECT` owned by `O.APPLET` such that either `O.JAVAOBJECT` is reachable from an applet instance distinct from `O.APPLET`, or `O.JAVAOBJECT` is reachable from a package P, or ([JCRE22], §8.5) `O.JAVAOBJECT` is remote reachable.

From Java Card platform, version 2.2.1, condition (2) becomes:

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

R.JAVA.15 ([JCRE22],§11.3.4.1, **Multiple Applet Instance Deletion**). The `S.ADEL` may perform `OP.DELETE_APPLET` upon several `O.APPLET` only if, (1) `S.ADEL` is currently selected, (2) every `O.APPLET` being deleted is deselected and (3) there is no `O.JAVAOBJECT` owned by any of the `O.APPLET` being deleted such that either `O.JAVAOBJECT` is reachable from an applet instance distinct from any of those `O.APPLET`, or

O.JAVAOBJECT is reachable from a package P, or ([JCRE22], §8.5) O.JAVAOBJECT is remote reachable.

From Java Card platform, version 2.2.1, condition (2) becomes:

(2') There is no instance of any of the O.APPLETs being deleted that is active in any logical channel.

R.JAVA.16 ([JCRE22], §11.3.4.2, **Applet/Library Package Deletion**). The S.ADEL may perform OP.DELETE_PCKG upon an O.CODE_PCKG only if, (1) S.ADEL is currently selected, (2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PCKG that is an instance of a class that belongs to O.CODE_PCKG exists on the card and (3) there is no package loaded on the card that depends on O.CODE_PCKG.

R.JAVA.17 ([JCRE22], §11.3.4.3, Applet **Package and Contained Instances Deletion**). The S.ADEL may perform OP.DELETE_PCKG_APPLET upon an O.CODE_PCKG only if, (1) S.ADEL is currently selected, (2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PCKG, which is an instance of a class that belongs to O.CODE_PCKG exists on the card, (3) there is no package loaded on the card that depends on O.CODE_PCKG and (4) for every O.APPLET of those being deleted it holds that: (i) O.APPLET is deselected and (ii) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance not being deleted, or O.JAVAOBJECT is reachable from a package not being deleted, or ([JCRE22],§8.5) O.JAVAOBJECT is remote reachable.

From Java Card platform, version 2.2.1, condition (4i) becomes:

(4i') There is no instance in the context of O.APPLET that is active in any logical channel.

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

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

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

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

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

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

FMT_SMR.1 SECURITY ROLES

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

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

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/ADEL The TSF shall be capable of performing the following security management functions: ***Modify theActiveApplets security attribute***.

5.1.4.2 Additional Deletion Requirements

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

FDP_RIP.1.1/ADEL The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation of the resource from*** the following objects: ***applet instances and/or packages when one of the deletion operations in FDP_ACC.2.1/ADEL is performed on them.***

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

Application note: There is no conflict with *FDP_ROL.1* requirements appearing in the document as of the bounds on the rollback: the deletion operation is out of the scope of the rollback (***FDP_ROL.1.1/FIREWALL***, p.75).

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

FPT_FLS.1.1/ADEL The TSF shall preserve a secure state when the following types of failures occur: *the applet deletion manager fails to delete a package/applet as described in* [JCRE22], §11.3.4.

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

Application note: The applet instance deletion must be atomic. The "secure state" refered to in the requirement must comply with the Java Card specifications. That is, if a reset or power fail occurs during the deletion process, then before any applet is selected in card, either the applet instance deletion is completed or the the applet shall be selectable and all objects owned by the applet remain unchanged (that is, the functionality of all applet instances on the card remains the same as prior to the unsuccessful deletion attempt) [JCRE22], §11.3.4.

5.1.5 *RMIG* Security Functional Requirements

This group is mainly devoted to specifying the policies that control the access to remote objects and the flow of information that takes place when the Java Card RMI service is used. There are specific control rules concerning the access to remote objects. The rules relate mainly to the lifetime of their corresponding remote references. Information concerning remote object references can be sent out of the card only if the corresponding remote object has been designated as exportable. Array parameters of remote method invocations are required to be allocated on the card as global arrays, the storage of references to those arrays must then be restricted as well.

5.1.5.1 Java Card RMI Policy

The *Java Card RMI* policy embodies both an access control and an information flow control policy.

FDP_ACC.2: COMPLETE ACCESS CONTROL

FDP_ACC.2.1/JCRMI The TSF shall enforce the ***JCRMI access control SFP*** on `S.CAD`, `S.JCRE`, `O.APPLET`, `O.REMOTE_OBJ`, `O.REMOTE_MTHD`, `O.ROR`, `O.RMI_SERVICE` and all operations among subjects and objects covered by the SFP.

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

`S.CAD`	The *CAD*. In the scope of this policy it represents the actor that requests, by issuing commands to the card, for Java Card RMI services.
`S.JCRE`	The *Java Card RE* is responsible on behalf of the card issuer of the bytecode execution and runtime environment functionalities. In the context of this security policy, the *Java Card RE* is in charge of the execution of the commands provided to (1) obtain the initial remote reference of an applet instance and (2) perform Remote Method Invocation.
`O.APPLET`	Any installed *applet*, its code and data.
`O.REMOTE_OBJ`	A remote object is an instance of a class that implements one (or more) remote interfaces. A remote interface is one that extends, directly or indirectly, the interface **`java.rmi.Remote`** ([JCAPI22]).
`O.ROR`	A remote object reference. It provides information concerning: (i) the identification of a remote object and (ii) the Implementation class of the object or the interfaces implemented by the class of the object. This is the object's information to which the CAD can access.

O.REMOTE_MTHD	A method of a remote interface.
O.RMI_SERVICE	These are instances of the class **javacardx.rmi.RMIService**. They are the objects that actually process the Java Card RMI services.

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

Operation	Description
OP.GET_ROR(O.APPLET,…)	Retrieves the initial remote object reference of a Java Card RMI based applet. This reference is the seed which the CAD client application needs to begin remote method invocations
OP.INVOKE(O.RMI_SERVICE,…)	Requests a remote method invocation on the remote object.

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

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

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

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

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

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

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

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

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

The **validity** of an *O.ROR* is defined in [JCRE22], §8.5.

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

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

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

R.JAVA.19 The `S.JCRE` may perform `OP.INVOKE` upon `O.RMI_SERVICE`, `O.ROR` and `O.REMOTE_MTHD`, only if, `O.ROR` is *valid* and belongs to the *Returned References* of `O.RMI_SERVICE`, and the *Identifier* of `O.REMOTE_MTHD` matches one of the remote methods in the class, indicated by the security attribute *class*, of the `O.REMOTE_OBJECT` to which `O.ROR` makes reference.

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

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

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

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

FDP_IFC.1 SUBSET INFORMATION FLOW CONTROL

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

Subjects[12] (prefixed with an "*S*") and information (prefixed with an "*I*") covered by this policy are:

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

A remote object reference descriptor provides information concerning: (i) the identification of the remote object and (ii) the implementation class of the object or the interfaces implemented by the class of the object. The descriptor is the only object's information to which the CAD can access.

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

There is a unique operation in this policy:

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

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 SIMPLE SECURITY ATTRIBUTES

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

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

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

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

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

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

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

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

FDP_IFF.1.4/JCRMI The TSF shall provide ***[assignment: list of additional SFP capabilities***].

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

FDP_IFF.1.6/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]***.

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

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

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

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

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

owner of the object may provoke the change of the security attribute value, the Java Card RE could perform the modification itself.

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

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

Application note: Remote objects' security attributes are created and initialized at the creation of the object, and except for the Exported attribute, the values of the attributes are not longer modifiable. The default value of the Exported attribute is true.

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

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

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

FMT_REV.1 REVOCATION

FMT_REV.1.1 **The TSF shall restrict the ability to revoke security attributes associated with the [selection:** *users, subjects, objects, [assignment: other additional resources]*] **to [assignment:** *the authorized identified roles*].

FMT_REV.1.1/JCRMI The TSF shall restrict the ability to revoke the **Returned References security attribute of an O.RMI_SERVICE** to **the Java Card RE** ***[assignment:*** *other authorized identified role*].

FMT_REV.1.2 **The TSF shall enforce the rules [assignment:** *specification of revocation rules*].

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

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

FMT_SMR.1 SECURITY ROLES

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

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

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

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

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

- ***Modify the security attribute Exported of an O.REMOTE_OBJ.***
- ***Modify the security attribute Returned References of an O.RMI_SERVICE.***

5.1.6 *LCG* Security Functional Requirements

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

5.1.6.1 Firewall Policy

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

FDP_ACC.2: COMPLETE ACCESS CONTROL

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

Subjects (prefixed with an "`S`"), objects (prefixed with an "`O`") and operations (prefixed with "OP") are exactly the same, which are covered by the ***FIREWALL*** access control SFP.

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

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

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

The following table describes the new security attribute attached to the subjects *S.PACKAGE*

Subject	Attributes
S.PACKAGE	Selection Status

The following table describes the possible values for the new security attributes.

Name	Description
Selection Status	Multiselectable, Non-multiselectable or "None"
ActiveApplets	List of package's *AID*s

The Java Card platform, version 2.2, introduces the possibility for an *applet* instance to be selected on multiple *logical channels* at the same time, or accepting other applets belonging to the same package being selected simultaneously. These applets are referred to as *multiselectable applet*s. *Applet*s 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 *package*s. No selection mode is defined for a library *package*.

Support for multiple *logical channels* (with multiple selected applet instances) requires a change to the *selected applet* concept as stated in Java Card System, version 2.1.1.. Since more than one applet instance can be selected at the same time, and one *applet* instance can be selected on different *logical channels* simultaneously, it is necessary to differentiate the state of the *applet* instances in more detail. An *applet* instance will be considered an *active applet instance* if it is currently selected in at least one logical channel.. An *applet* instance is the *currently selected applet instance* only if it is processing the current command. There can only be one currently selected *applet* instance at a given time. ([JCRE22],§4).

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

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

The same rules of the ***FIREWALL SFP*** defined in 5.1.1.1 except for rule **R.JAVA.4**, which must be replaced by the following rule:

R.JAVA.20 ([JCRE22], §6.2.8.6,) An *S.PACKAGE* may perform *OP.INVK_INTERFACE* upon an *O.JAVAOBJECT* whose Sharing attribute has the value **"SIO"**, and whose Context attribute has the value "*Package AID*", only if one of the following applies:

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

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

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

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

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

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

5.1.6.2 Additional Requirements on Logical Channels

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

The following element must substitute former ***FDP_RIP.1.1/TRANSIENT***:

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

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

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

5.1.7 *ODELG* Security Functional Requirements

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

FDP_RIP.1 SUBSET RESIDUAL INFORMATION PROTECTION

FDP_RIP.1.1/ODEL The TSF shall ensure that any previous information content of a resource is made unavailable upon the ***de-allocation of the resource from*** the following objects: ***the objects owned by the context of an applet instance which triggered the execution of the method*** **`javacard.framework.JCSystem.requestObjectDeletion()`**.

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

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

FPT_FLS.1 FAILURE WITH PRESERVATION OF SECURE STATE

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

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

5.1.8 *CarG* Security Functional Requirements

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

FCO_NRO.2 ENFORCED PROOF OF ORIGIN

FCO_NRO.2.1 **The TSF shall enforce the generation of evidence of origin for transmitted [assignment: list of information types] at all times.**

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

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

FCO_NRO.2.2 **The TSF shall be able to relate the [assignment: list of attributes] of the originator of the information, and the [assignment: list of information fields] of the information to which the evidence applies.**

FCO_NRO.2.2/CM The TSF shall be able to relate the ***identity*** of the originator of the information, and the *application package* ***contained in the*** information to which the evidence applies.

FCO_NRO.2.3 **The TSF shall provide a capability to verify the evidence of origin of information to [selection: originator, recipient, [assignment: list of third parties]] given [assignment: limitations on the evidence of origin].**

FCO_NRO.2.3/CM The TSF shall provide a capability to verify the evidence of origin of information to ***the recipient*** given **[assignment: limitations on the evidence of origin].**

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

FIA_UID.1 TIMING OF IDENTIFICATION

FIA_UID.1.1/CM The TSF shall allow ***[assignment: list of TSF-mediated actions]*** on behalf of the user to be performed before the user is identified.

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

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

FDP_IFC.2 COMPLETE INFORMATION FLOW CONTROL

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

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

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

The operations (prefixed with "*OP*") that make information to flow between the subjects are those enabling to send a message through and to receive a message from the communication channel linking the card to the outside world. It is assumed that an attacker is able to read any message sent through the channel as clear text. Moreover, the attacker may capture any message sent through the communication channel and send its own messages to the other subjects.

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

The information (prefixed with an "**I**") controlled by the typing policy is the *APDUs* exchanged by the subjects through the communication channel linking the card and the CAD. Each of those messages contain part of an application package that is required to be loaded on the card, as well as any control information used by the subjects (either ***S.BCV*** or ***S.SPY)*** in the communication protocol.

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

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

FDP_IFF.1 SIMPLE SECURITY ATTRIBUTES

FDP_IFF.1.1/CM The TSF shall enforce the ***PACKAGE LOADING information flow control SFP*** based on the following types of subject and information security attributes: ***[assignment: the minimum number and type of security attributes].***

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

FDP_IFF.1.2/CM The TSF shall permit an information flow between a controlled subject and controlled information through a controlled operation if the following rules hold: ***[assignment: the rules describing the communication protocol used by the CAD and the card for transmitting a new package].***

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

An example of such a communication protocol can be found in Appendix D of [GP].

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

FDP_IFF.1.4/CM The TSF shall provide ***[assignment: list of additional SFP capabilities]***.

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

FDP_IFF.1.6/CM The TSF shall explicitly deny an information flow based on the following rules: ***[assignment: other rules, based on security attributes, that explicitly deny information flows]***

FDP_UIT.1 DATA EXCHANGE INTEGRITY

These requirements apply to those configurations where bytecode verification is not considered as being part of the TOE. If this is the case, then the bytecode verifier can be seen as an external IT product, and *packages* to be loaded on the card are user data in transit from that external product to the *Java Card System*.

FDP_UIT.1.1 **The TSF shall enforce the [assignment: access control SFP(s) and/or information flow control SFP(s)] to be able to [selection: transmit, receive] user data in a manner protected from [selection: modification, deletion, insertion, replay] errors.**

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

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

FDP_UIT.1.2 **The TSF shall be able to determine on receipt of user data, whether [selection: modification, deletion, insertion, replay] has occurred.**

FDP_UIT.1.2/CM The TSF shall be able to determine on receipt of user data, whether ***modification, deletion, insertion, replay of some of the pieces of the application sent by the CAD*** has occurred.

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/CM The TSF shall enforce the ***PACKAGE LOADING information flow control SFP*** to restrict the ability to [selection: *change default, query, modify, delete,* [assignment: *other operations*]] the security attributes [assignment: *list of security attributes*] to [assignment: *the authorized identified roles*].

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

FMT_MSA.3.2/CM The TSF shall allow the [assignment: *the authorized identified roles*] to specify alternative initial values to override the default values when an object or information is created.

FMT_SMR.1 Security roles

FMT_SMR.1.1/CM The TSF shall maintain the roles: [assignment: *the authorized identified roles*].

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

FMT_SMF.1 Specification of Management Functions

FMT_SMF.1.1/CM The TSF shall be capable of performing the following security management functions: [assignment: *list of security management functions to be provided by the TSF*].

FTP_ITC.1 Inter-TSF trusted channel

The following requirements apply to those configurations where bytecode verification is not considered as being part of the TOE. If this is the case, then the *CAD* can be seen as a remote trusted IT product, and *package*s to be loaded on the card shall be transmitted using an inter-TSF trusted channel to prevent them from being modified during downloading. Such trusted channel connects the embedded *Java Card System* to the secured environment where the package has been verified.

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

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

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

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

FTP_ITC.1.3 **The TSF shall initiate communication via the trusted channel for [assignment: *list of functions for which a trusted channel is required*].**

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

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

5.1.9 *SCPG* Security Functional Requirements

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

UNDERLYING ABSTRACT MACHINE TEST (FPT_AMT)

FPT_AMT.1.1 **The TSF shall run a suite of tests [selection: *during initial start-up, periodically during normal operation, at the request of an authorized user, [assignement: other conditions]*] to demonstrate the correct operation of the security assumptions provided by the abstract machine that underlies the TSF.**

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

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

FAIL SECURE (FPT_FLS)

FPT_FLS.1.1/SCP The TSF shall preserve a secure state when the following types of failures occur: [assignment: *list of types of failures in the TSF*].

FAULT TOLERANCE (FRU_FLT)

FRU_FLT.1.1/SCP The TSF shall ensure the operation of [assignment: *list of TOE capabilities*] when the following failures occur: [assignment: *list of type of failures*].

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

TSF PHYSICAL PROTECTION (FPT_PHP)

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

DOMAIN SEPARATION (FPT_SEP)

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

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

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

REFERENCE MEDIATION (FPT_RVM)

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

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

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

TRUSTED RECOVERY (FPT_RCV)

FPT_RCV.3.1/SCP When automated recovery from [assignment: *list of failures/service discontinuities*] is not possible, the TSF shall enter a maintenance mode where the ability to return the TOE to a secure state is provided.

FPT_RCV.3.2/SCP For [assignment: *list of failures/service discontinuities*], the TSF shall ensure the return of the TOE to a secure state using automated procedures.

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

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

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

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

5.1.10 *CMGRG* Security Functional Requirements

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

The security requirements below helps defining a policy for controlling access to card content management operations and for expressing card issuer security concerns. This policy shall be highly dependent on the particular security and card management architecture present in the card. Therefore the policy should be accordingly refined when developing conformant Security Targets.

FDP_ACC.1 SUBSET ACCESS CONTROL

FDP_ACC.1.1 **The TSF shall enforce the [assignment: access control SFP] on [assignment: list of subjects, objects, and operations among subjects and objects covered by the *SFP*].**

FDP_ACC.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** on [assignment: list of subjects, objects, and operations among subjects and objects covered by the *SFP*].

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

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

FDP_ACF.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to objects based on the following: [assignment: *list of subjects and objects controlled under the indicated SFP, and for each, the SFP-relevant security attributes, or named groups of SFP-relevant security attributes*].

FDP_ACF.1.2/CMGR The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [assignment: *rules governing access among controlled subjects and controlled objects using controlled operations on controlled objects*].

FDP_ACF.1.3/CMGR The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [assignment: *rules, based on security attributes, that explicitly authorize access of subjects to objects*].

FDP_ACF.1.4/CMGR The TSF shall explicitly deny access of subjects to objects based on the [assignment: *rules, based on security attributes, that explicitly deny access of subjects to objects*].

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/CMGR The TSF shall enforce the ***CARD CONTENT MANAGEMENT access control SFP*** to restrict the ability to [selection: *change default, query, modify,*

delete, [assignment: *other operations*]] the security attributes [assignment: *list of security attributes*] to [assignment: *the authorized identified roles*].

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

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

FMT_MSA.3.2/CMGR The TSF shall allow the [assignment: *the authorized identified roles*] to specify alternative initial values to override the default values when an object or information is created.

FMT_SMR.1 SECURITY ROLES

FMT_SMR.1.1/CMGR The TSF shall maintain the roles: [assignment: *the authorized identified roles*].

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

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/CMGR **The TSF shall be capable of performing the following security management functions: [assignment: *list of security management functions to be provided by the TSF*].**

FIA_UID.1 TIMING OF IDENTIFICATION

FIA_UID.1.1/CMGR The TSF shall allow [assignment***: list of TSF-mediated actions***] on behalf of the user to be performed before the user is identified.

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

Application note: The list of TSF-mediated actions depends on the particular card manager security architecture implemented, but typically card content modification requires for the user attempting the modification to be identified. Here by user is meant the one(s) that in the Security Target shall be associated to the role(s) defined in the component *FMT_SMR.1/CMGR*

5.1.11EMG Security Functional Requirements

This group contains the security requirements for the management of the external memory, introduced in the version 2.2.2 of the Java Card System (cf. [JCAPI222], optional package javacardx.external).

5.1.11.1 External Memory Policy

FDP_ACC.1: SUBSET ACCESS CONTROL

FDP_ACC.1.1/EXT_MEM The TSF shall enforce the ***EXTERNAL MEMORY access control SFP*** on subject *S.APPLET*, ***object*** *O.EXT_MEM_INSTANCE*, and operations *OP.CREATE_EXT_MEM_INSTANCE*, *OP.READ_EXT_MEM* and *OP.WRITE_EXT_MEM*.

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

Subject/Object	Description
S.APPLET	Any applet instance
O.EXT_MEM_INSTANCE	Any External Memory Instance created from the MemoryAccess Interface of the Java Card API [JCAPI222].

The following table describes the operations (prefixed with "*OP*") of this policy. Each operation has a specific number of parameters given between brackets, among which there is the "**accessed object**", the first one, when applicable.

Operation	Description
OP.CREATE_EXT_MEM_INSTANCE	Creation of an instance of the MemoryAccess Interface.
OP.READ_EXT_MEM(O.EXT_MEM_INSTANCE, address*)*	Reading the external memory
OP.WRITE_EXT_MEM(O.EXT_MEM_INSTANCE, address*)*	Writing the external memory

FDP_ACF.1 SECURITY ATTRIBUTE BASED ACCESS CONTROL

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

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

Object	Security Attribute	Description
O.EXT_MEM_INSTANCE	Address space	Accessible memory portion.

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

R.JAVA.21 ([JCAPI222]) Any subject S.APPLET that performs OP.CREATE_EXT_MEM_INSTANCE obtains an object O.EXT_MEM_INSTANCE that addresses a memory space different from that of the Java Card System.

Application note: The actual mechanism for creating an instance of external memory is implementation-dependent. This rule only says that the accessible address space must not interfere with that of the Java Card System.

R.JAVA.22 ([JCAPI222]) Any subject S.APPLET may perform OP.READ_EXT_MEM(*O.EXT_MEM_INSTANCE*, address) provided the address belongs to the space of the O.EXT_MEM_INSTANCE.

R.JAVA.23 ([JCAPI222]) Any subject S.APPLET may perform OP.WRITE_EXT_MEM(*O.EXT_MEM_INSTANCE*, address) provided the address belongs to the space of the O.EXT_MEM_INSTANCE.

Application note: The creation of and the access to an external memory instance is subject to the Firewall rules.

FDP_ACF.1.3/EXT_MEM The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: **[assignment:** *rules, based on security attributes, that explicitly authorize access of subjects to objects*]**.**

FDP_ACF.1.4/EXT_MEM The TSF shall explicitly deny access of subjects to objects based on the **[assignment:** ***rules, based on security attributes, that explicitly deny access of subjects to objects***]**.**

FMT_MSA.1 MANAGEMENT OF SECURITY ATTRIBUTES

FMT_MSA.1.1/EXT_MEM The TSF shall enforce the EXTERNAL MEMORY ***access control SFP*** to restrict the ability to **set up the** security attribute *address space* to **the Java Card RE.**

FMT_MSA.3 STATIC ATTRIBUTE INITIALIZATION

FMT_MSA.3.1/EXT_MEM The TSF shall enforce the *EXT_MEM* ***access control SFP*** to provide ***no*** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/EXT_MEM The TSF shall allow the Java Card RE to specify alternative initial values to override the default values when an object or information is created.

Application note: Upon creation of an external memory instance, the Java Card RE gets the address space value for the newly created object. This is implementation dependent.

FMT_SMF.1 SPECIFICATION OF MANAGEMENT FUNCTIONS

FMT_SMF.1.1/EXT_MEM The TSF shall be capable of performing the following security management functions: ***Set up the address space security attribute.***

5.2 TOE SECURITY ASSURANCE REQUIREMENTS

The assurance requirement of the Protection Profiles is **EAL 4 augmented**.

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

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

ACM_AUT.1 Partial Configuration Management automation

ACM_CAP.4 Generation support and acceptance procedures

ACM_SCP.2 Problem tracking Configuration Management coverage

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

ADV_FSP.2 Fully defined external interfaces

ADV_HLD.2 Security enforcing high-level design

ADV_LLD.1 Descriptive low-level design

ADV_RCR.1 Informal correspondence demonstration

ADV_SPM.1 Informal TOE security policy model

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

ADO_DEL.2 Detection of modification

This SAR ensures the integrity of the TOE and its documentation during the transfer of the TOE between all the actors appearing in the first two stages. Procedures shall ensure protection of TOE material/information under delivery and storage that corrective actions are taken in case of improper

operation in the delivery process and storage and that people dealing with the procedure for delivery have the required skills.

ADO_IGS.1 Installation, generation, and start-up procedures

AGD_ADM.1 Administrator guidance

AGD_USR.1 User guidance

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

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

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

ALC_DVS.1 Identification of security measures

ALC_LCD.1 Developer defined life-cycle model

ALC_TAT.1 Well-defined development tools

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

ATE_COV.2 Analysis of Coverage

ATE_DPT.1 Testing: high-level design

ATE_FUN.1 Functional testing

ATE_IND.2 Independent testing - sample

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

AVA_MSU.2 Validation of analysis

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

AVA_SOF.1 Strength of TOE security function evaluation

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

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

AVA_VLA.3 Moderately resistant

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

ADV_IMP.2 Implementation of the TSF.

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

6 Rationale

This chapter presents the evidence used in the evaluation of the Protection Profiles. This evidence supports the claims that each of them is a complete and cohesive set of requirements and that a conformant TOE would provide an effective set of IT security countermeasures within the security environment.

6.1 SECURITY OBJECTIVES RATIONALE

This section demonstrates that the stated security objectives address the entire security environment aspects identified. Each security objective is correlated to at least one threat, organizational security policy or assumption.

The section is structured by configuration and then by the type of rationale.

6.1.1 Minimal Configuration

6.1.1.1 Threats Related to Security Objectives

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

T.PHYSICAL	Covered by *OE.1215H*SCP*.IC*. Physical protections rely on the underlying platform and are therefore an environmental issue.

CONFIDENTIALITY & INTEGRITY

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

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

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

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

As the firewall is a software tool automating critical controls, the objective *O.1152H*ALARM asks for it to provide clear warning and error messages, so that the appropriate counter-measure can be taken.

Concerning the confidentiality and integrity of application sensitive data, as *applet*s may need to share some data or communicate with the *CAD*, cryptographic functions are required to actually protect the exchanged information (*O.CIPHER*). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the *applet*s to use them. Keys and PIN's are particular cases of an application's sensitive data[13] 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 shared resource of all the applications. The disclosure of such kind of data is prevented by the (*O.SHRD_VAR_CONFID*) security objective. The integrity of the information stored in that buffer is ensured by the (*O.SHRD_VAR_INTEG*) objective.

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

IDENTITY USURPATION

T.SID.1

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

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

T.SID.2

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

UNAUTHORIZED EXECUTIONS

T.EXE-CODE.1

Unauthorized execution of a method is prevented by the objective *OE.VERIFICATION*. This threat particularly concerns the point (8) of the security issue (access modifiers and scope of visibility for classes, fields

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

and methods). The *O.FIREWALL* objective is also concerned, because it prevents the execution of non-shareable methods of a class instance by any subject apart from the class instance owner.

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

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

DENIAL OF SERVICE

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

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

The objective *OE.CARD-MANAGEMENT* supports *OE.1224HVERIFICATION* and contributes to cover all the threats on confidentiality and integrity of code and data. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.1070H*SID*.1*.

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

	OE.NO-INSTALL	OE.VERIFICATION	OE.CARD-MANAGEMENT	O.SHRD_VAR_INTEG	O.SHRD_VAR_CONFID	O.1145HFIREWALL	O.1148HNATIVE	O.1139HOPERATE	O.ALARM	O.REALLOCATION	O.1142HRESOURCES	O.1138HSID	OE.1215HSCP.IC	OE.SCP.RECOVERY	OE.SCP.SUPPORT	O.1158HCIPHER	O.KEY-MNGT	O.PIN-MNGT	O.TRANSACTION
T.PHYSICAL													X						
T.1047HCONFID-JCS-CODE T.1057HINTEG-APPLI-CODE T.1060HINTEG-JCS-CODE		X	X																
T.1054HCONFID-JCS-DATA T.1067HINTEG-JCS-DATA		X	X			X		X	X			X		X	X				
T.1050HCONFID-APPLI-DATA		X	X		X	X		X	X	X		X		X	X	X	X	X	X
T.1063HINTEG-APPLI-		X	X	X		X		X	X	X		X		X	X	X	X	X	X
T.SID.1	X		X			X						X							
T.SID.2						X		X				X		X	X				
T.EXE-CODE.1		X				X													
T.EXE-CODE.2		X																	
T.NATIVE		X					X												
T.RESOURCES								X			X			X	X				

Table 3: Minimal **Configuration threats rationale**

6.1.1.2 Assumptions Related to Security Objectives

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

In this configuration all the security objectives directly or indirectly depend on the behavior of the native code embedded on the card. This trusted native code is not subject to change during the lifetime of the card. The objective *OE.NATIVE* ensures that the environmental assumption *A.1023HNATIVE* is upheld. The objective *OE.VERIFICATION* upholds the assumption *A.VERIFICATION*.

The assumptions *A.NO-DELETION* and *A.NO-INSTALL* are also upheld by the environmental objective *OE.CARD-MANAGEMENT*.

Table 4 provides a mapping of security objectives to the assumptions made on the environment of the TOE.

	OE.CARD-MANAGEMENT	OE.NATIVE	OE.NO-DELETION	OE.NO-INSTALL	OE.VERIFICATION
A.1023HNATIVE		X			
A.NO-DELETION	X		X		
A.NO-INSTALL	X			X	
A.VERIFICATION					X

Table 4: Minimal **Configuration assumptions rationale**

The following security objectives of the TOE are related to the assumptions made for this configuration as follows:

O.FIREWALL The controlled sharing of data owned by different applications assumes that the code of the applications is well typed (*A.VERIFICATION*). Secured installation ensures the correct initialization of TSF data such as the identity of the applications (*A.NO-INSTALL*).

O.SID The correct identification of the applications depends on the assumptions stating that pre-issuance applications have been correctly installed (*A.NO-INSTALL*), and that those are exactly the applications that will be on the card (*A.NO-DELETION*).

6.1.1.3 Organizational Policies Related to Security Objectives

No organizational security policy has been defined for this configuration.

6.1.2 Java Card System Standard 2.1.1 Configuration

6.1.2.1 Threats Related to Security Objectives

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

T.PHYSICAL Covered by *OE.1215H*SCP.*IC*. Physical protections rely on the underlying platform and are therefore an environmental issue.

CONFIDENTIALITY & INTEGRITY

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

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

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

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

As the firewall is a software tool automating critical controls, the objective *O.1152H*ALARM asks for it to provide clear warning and error messages, so that the appropriate counter-measure can be taken.

Concerning the confidentiality and integrity of application sensitive data, as *applet*s may need to share some data or communicate with the *CAD*, cryptographic functions are required to actually protect the exchanged information (*O.CIPHER*). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the *applet*s to use them. Keys and PIN's are particular cases of an application's sensitive data[14] that ask for appropriate management (*O.KEY-MNGT*, *O.PIN-MNGT*, *O.TRANSACTION*). If the PIN class of the Java Card API is used, the objective (*O.FIREWALL*) is also concerned.

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

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

IDENTITY USURPATION

T.SID.1

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

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

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

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

countered by the objective (*O.SHRD_VAR_CONFID*) and (*O.SHRD_VAR_INTEG*).

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

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

UNAUTHORIZED EXECUTIONS

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

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

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

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

DENIAL OF SERVICE

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

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

Note that, for what relates to CPU usage, the Java Card platform is single–threaded and it is possible for an ill–formed application (either native or not) to monopolize the CPU. However, a smart card can be physically interrupted (card removal or hardware reset) and most CAD 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.

MODIFICATIONS OF THE SET OF APPLICATIONS

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

INTEGRITY AND INSTALLATION

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

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

The objective *OE.CARD-MANAGEMENT* supports *OE.1224HVERIFICATION* and contributes to cover all the threats on confidentiality and integrity of code and data, the *T.1097H*INSTALL threat, and the *T.INTEG-APPLI-CODE.2* and *T.INTEG-APPLI-DATA.2* threats. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.1070H*SID.*1*.

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

	O.INSTALL	*O.LOAD*	*OE.VERIFICATION*	*OE.CARD-MANAGEMENT*	*OE.APPLET*	*O.SHRD_VAR_INTEG*	*O.SHRD_VAR_CONFID*	*O.1145HFIREWALL*	*O.1148HNATIVE*	*O.1139HOPERATE*	*O.ALARM*	*O.REALLOCATION*	*O.1142HRESOURCES*	*O.1138HSID*	*OE.1215HSCP.IC*	*OE.SCP.RECOVERY*	*OE.SCP.SUPPORT*	*O.1158HCIPHER*	*O.KEY-MNGT*	*O.PIN-MNGT*	*O.TRANSACTION*
T.PHYSICAL															X						
*T.1047H*CONFID-JCS-CODE																					
*T.1057H*INTEG-APPLI-CODE			X	X																	
*T.1060H*INTEG-JCS-CODE																					

*T.1054H*CONFID-JCS-DATA *T.1067H*INTEG-JCS-DATA			X	X				X		X	X			X		X	X				
*T.1050H*CONFID-APPLI-DATA			X	X			X	X		X	X	X		X		X	X	X	X	X	X
*T.1063H*INTEG-APPLI-			X	X		X		X		X	X	X		X		X	X	X	X	X	X
T.SID.1	X			X		X	X	X						X							
T.SID.2	X							X		X				X		X	X				
T.EXE-CODE.1			X					X													
T.EXE-CODE.2			X																		
T.NATIVE			X		X				X												
T.RESOURCES	X									X			X			X	X				
T.INSTALL	X	X		X																	
T.INTEG-APPLI-CODE.2		X		X																	
T.INTEG-APPLI-DATA.2		X		X																	

Table 5: Java Card System Standard 2.1.1 **Configuration threats rationale**

6.1.2.2 Assumptions Related to Security Objectives

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

In this configuration all the security objectives directly or indirectly depend on the behavior of the native code embedded on the card. This trusted native code is not subject to change during the lifetime of the card. The objective *OE.NATIVE* ensures that the environmental assumption *A.1023HNATIVE* is upheld. The objective *OE.APPLET* covers the assumption *A.APPLET*, and contributes to the enforcement of the objective *O.NATIVE* in the presence of post-issuance downloaded applications. The objective *OE.VERIFICATION* upholds the assumption *A.VERIFICATION*.

Table 6 provides a mapping of security objectives to the assumptions made on the environment of the TOE.

	OE.NATIVE	*OE.APPLET*	*OE.CARD-MANAGEMENT*	*OE.VERIFICATION*
A.NATIVE	X			
A.APPLET		X		
A.DELETION			X	
A.VERIFICATION				X

Table 6: Java Card System Standard 2.1.1 **Configuration assumptions rationale**

The assumption *A.DELETION* is upheld by the environmental objective *OE.CARD-MANAGEMENT*.

6.1.2.3 Organizational Policies Related to Security Objectives

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

6.1.3 Java Card System Standard 2.2 Configuration

6.1.3.1 Threats Related to Security Objectives

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

T.PHYSICAL COVERED BY *OE.1215H*SCP.*IC*. PHYSICAL PROTECTIONS RELY ON THE UNDERLYING PLATFORM AND ARE THEREFORE AN ENVIRONMENTAL ISSUE.CONFIDENTIALITY & INTEGRITY

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

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

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

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

As the firewall is a software tool automating critical controls, the objective *O.1152H*ALARM asks for it to provide clear warning and error messages, so that the appropriate counter-measure can be taken.

Concerning the confidentiality and integrity of application sensitive data, as *applet*s may need to share some data or communicate with the *CAD*, cryptographic functions are required to actually protect the exchanged information (*O.CIPHER*). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the *applet*s to use them. Keys, PINs and biometric templates are particular cases of an application's sensitive data[15] that ask for appropriate management (*O.KEY-MNGT*, *O.PIN-MNGT*, *O.BIO*-MNGT, *O.TRANSACTION*). If the PIN class of the Java Card API is used, the objective (*O.FIREWALL*) is also concerned.

---

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

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

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.*O.EXT_MEM* counters unauthorized disclosing and modification of Java Card System data through the use of the external memory features of the TOE.

IDENTITY USURPATION

T.SID.1

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

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

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

T.SID.2

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

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

UNAUTHORIZED EXECUTIONS

T.EXE-CODE.1

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

T.EXE-CODE.2

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

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

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

DENIAL OF SERVICE

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

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

Note that, for what relates to CPU usage, the Java Card platform is single–threaded and it is possible for an ill–formed application (either native or not) to monopolize the CPU. However, a smart card can be physically interrupted (card removal, RF signal loss 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.

MODIFICATIONS OF THE SET OF APPLICATIONS

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

INTEGRITY AND INSTALLATION

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

T.INTEG-APPLI-DATA.2 The attacker modifies (part of) the initialization data contained in an application *package* when the package is transmitted to the card for

installation. In this configuration the integrity of a package's code is covered by the objective *O.LOAD*.

UNAUTHORIZED EXECUTIONS

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

CARD MANAGEMENT

T.DELETION This threat is covered by the *O.1186HDELETION* security objective.

OBJECT DELETION

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

The objective *OE.CARD-MANAGEMENT* supports *OE.1224HVERIFICATION* and contributes to cover all the threats on confidentiality and integrity of code and data, the *T.1097H*INSTALL threat, the *T.1113HDELETION* threat and the *T.INTEG-APPLI-CODE.2* and *T.INTEG-APPLI-DATA.2* threats. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.1070H*SID.*1*.

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

	O.INSTALL	O.LOAD	OE.VERIFICATION	OE.CARD-MANAGEMENT	OE.APPLET	O.SHRD_VAR_INTEG	O.SHRD_VAR_CONFID	O.1145HFIREWALL	O.1148HNATIVE	O.1139HOPERATE	O.ALARM	O.REALLOCATION	O.1142HRESOURCES	O.1138HSID	OE.1215HSCP.IC	OE.SCP.RECOVERY	OE.SCP.SUPPORT	O.1158HCIPHER	O.KEY-MNGT	O.PIN-MNGT	O.BIO-MNGT	O.TRANSACTION	O.1186HDELETION	O.REMOTE	O.OBJ-DELETION	O.EXT_MEM
T.PHYSICAL															X											
T.1047HCONFID-JCS-CODE T.1057HINTEG-APPLI-CODE T.1060HINTEG-JCS-CODE			X	X																						X
T.1054HCONFID-JCS-DATA T.1067HINTEG-JCS-DATA			X	X				X		X	X			X		X	X									X
T.1050HCONFID-APPLI-			X	X			X	X		X	X	X		X		X	X	X	X	X	X	X				X
T.1063HINTEG-APPLI-			X	X		X		X		X	X	X		X		X	X	X	X	X	X	X				X
T.1070HSID.1	X			X		X	X	X						X												
T.1073HSID.2	X							X		X				X		X	X									
T.EXE-CODE.1			X					X																		
T.EXE-CODE.2			X																							
T.1084HNATIVE			X		X				X																	
T.1087HRESOURCES	X									X			X			X	X									
T.1097HINSTALL	X	X		X																						
T.INTEG-APPLI-CODE.2		X		X																						
T.INTEG-APPLI-DATA.2		X		X																						
T.1113HDELETION				X																			X			
T.EXE-CODE-REMOTE																								X		
T.1116HOBJ-DELETION																									X	

Table 7: Java Card System Standard 2.2 **Configuration threats rationale**

6.1.3.2 Assumptions Related to Security Objectives

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

In this configuration all the security objectives directly or indirectly depend on the behavior of the native code embedded on the card. This trusted native code is not subject to change during the lifetime of the card. The objective *OE.NATIVE* ensures that the environmental assumption *A.1023HNATIVE* is upheld. The objective *OE.APPLET* covers the assumption *A.APPLET*, and contributes to the enforcement of the objective *O.NATIVE* in the presence of post-issuance downloaded applications. The objective *OE.VERIFICATION* upholds the assumption *A.VERIFICATION*.

Table 8 provides a mapping of security objectives to the assumptions made on the environment of the TOE.

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

Table 8: Java Card System Standard 2.2 **Configuration assumptions rationale**

6.1.3.3 Organizational Policies Related to Security Objectives

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

6.1.4 Defensive Configuration

6.1.4.1 Threats Related to Security Objectives

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

T.PHYSICAL Covered by *OE.1215H*SCP.*IC*. Physical protections rely on the underlying platform and are therefore an environmental issue.

CONFIDENTIALITY & INTEGRITY

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

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

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

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

As both the bytecode verifier and the firewall are software tools automating critical controls, the objective *O.1152H*ALARM asks for them to provide clear warning and error messages, so that the appropriate counter-measure can be taken.

Concerning the confidentiality and integrity of application sensitive data, as *applet*s may need to share some data or communicate with the *CAD*, cryptographic functions are required to actually protect the exchanged information (*O.CIPHER*). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the *applet*s to use them. Keys and PIN's are particular cases of an application's sensitive data[16] that ask for appropriate management (*O.KEY-MNGT*, *O.PIN-MNGT*, *O.BIO*-MNGT, *O.TRANSACTION*). If the PIN class of the Java Card API is used, the objective (*O.FIREWALL*) is also concerned.

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

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

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. *O.EXT_MEM* counters unauthorized disclosing and modification of Java Card System data through the use of the external memory features of the TOE.

IDENTITY USURPATION

T.SID.1

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

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

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

T.SID.2

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

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

UNAUTHORIZED EXECUTIONS

T.EXE-CODE.1

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

T.EXE-CODE.2

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

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

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

DENIAL OF SERVICE

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

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

Note that, for what relates to CPU usage, the Java Card platform is single–threaded and it is possible for an ill–formed application (either native or not) to monopolize the CPU. However, a smart card can be physically interrupted (card removal or hardware reset) and most CAD 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.

MODIFICATIONS OF THE SET OF APPLICATIONS

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

UNAUTHORIZED EXECUTIONS

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

CARD MANAGEMENT

T.DELETION This threat is covered by the *O.1186HDELETION* security objective.

OBJECT DELETION

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

The objective *OE.CARD-MANAGEMENT* contributes to cover the threats *T.1097H*INSTALL and *T.1113HDELETION*. The objective also contributes, by preventing usurpation of identity resulting from a malicious installation of an applet on the card, to counter the threat *T.1070H*SID.*1*.

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

	O.INSTALL	*O.VERIFICATION*	*OE.CARD-MANAGEMENT*	*OE.APPLET*	*O.SHRD_VAR_INTEG*	*O.SHRD_VAR_CONFID*	*O.1145HFIREWALL*	*O.1148HNATIVE*	*O.1139HOPERATE*	*O.ALARM*	*O.REALLOCATION*	*O.1142HRESOURCES*	*O.1138HSID*	*OE.1215HSCP.IC*	*OE.SCP.RECOVERY*	*OE.SCP.SUPPORT*	*O.1158HCIPHER*	*O.KEY-MNGT*	*O.PIN-MNGT*	*O.BIO*-MNGT	*O.TRANSACTION*	*O.1186HDELETION*	*O.REMOTE*	*O.OBJ-DELETION*	*O.EXT_MEM*
T.PHYSICAL														X											
*T.1047H*CONFID-JCS-CODE *T.1057H*INTEG-APPLI-CODE *T.1060H*INTEG-JCS-CODE		X																							X
*T.1054H*CONFID-JCS-DATA *T.1067H*INTEG-JCS-DATA		X					X		X	X			X		X	X									X
*T.1050H*CONFID-APPLI-		X				X	X		X	X	X		X		X	X	X	X	X	X	X				X
*T.1063H*INTEG-APPLI-		X			X		X		X	X	X		X		X	X	X	X	X	X	X				X
T.SID.1	X		X		X	X	X						X												
T.SID.2	X						X		X				X		X	X									
T.EXE-CODE.1		X					X																		
T.EXE-CODE.2		X																							
T.NATIVE		X		X				X																	
T.RESOURCES	X								X			X			X	X									
T.INSTALL	X		X																						
T.DELETION			X																			X			
T.EXE-CODE-REMOTE																							X		
T.OBJ-DELETION																								X	

Table 9: Defensive **Configuration threats rationale**

6.1.4.2 Assumptions Related to Security Objectives

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

In this configuration all the security objectives directly or indirectly depend on the behavior of the native code (*A.1023HNATIVE*) embedded on the card. This trusted native code is not subject to change during the lifetime of the card.

Table 10 provides a mapping of security objectives to the assumptions made on the environment of the TOE.

	OE.NATIVE
A.NATIVE	X

Table 10: Defensive **Configuration assumptions rationale**

6.1.4.3 Organizational Policies Related to Security Objectives

No organizational security policy has been defined for this configuration.

6.2 SECURITY REQUIREMENTS RATIONALE

This section is devoted to demonstrate that the set of security requirements (both on the TOE and on the environment) is suitable to meet security objectives. The presentation follows the same structure as §4.1, listing the requirements that are related to each objective of each configuration.

The following conventions shall are used throughout this section:

> In the text of the rationales there shall be explicit references to (access and information flow) control policies, as contributing to meet certain security objectives. These references shall be associated to the principal security components by means of which those policies are defined, FDP_ACC and FDP_ACF in the case of control policies; FDP_IFC and FDP_IFF in the case of information flow ones, as well as to all the SFRs on which the afore mentioned components depend. The rationale tables, on the contrary, shall list the security objectives that the policies' components contribute to meet.
>
> The name of a SFR class component shall be used to make reference to (all) the iterations of that component that are present in a configuration. By present in a configuration it must be understood as belonging to one of the groups included in that configuration.
>
> A reference to a particular iteration of a SFR component shall be denoted as Component_Name/Label, where Label shall be the name of the TOE component.

6.2.1 Minimal Configuration

6.2.1.1 TOE Security Requirements Rationale

Unless explicitly stated, all the security functional requirements to which this section makes reference are those specified in the group *CoreG* (§5.1.1).

IDENTIFICATION

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

EXECUTION

O.OPERATE	The TOE is protected in various ways against *applet*s' actions (***FPT_RVM.1***, ***FPT_SEP.1***, *FPT_TDC.1*), the ***FIREWALL*** *access control policy* (*FDP_ACC.2*,

FDP_ACF.1), and is able to detect and block various failures or security violations during usual working (*FPT_FLS.1*, *FAU_ARP.1*). Startup of the TOE is covered by *FPT_TST.1*, and indirectly by *FPT_AMT.1* (this latter defined in group *SCPG* §5.1.9).

Its security-critical parts and procedures are also protected: communication with external users and their internal subjects is well controlled (*FIA_ATD.1*, ***FIA_USB.1***) to prevent alteration of TSF data (also protected by components of the FPT class).

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

O.RESOURCES

The TSFs detects stack/memory overflows during execution of applications (*FAU_ARP.1*, *FPT_FLS.1*). Memory management is controlled by the TSF (***FMT_MTD.1***, ***FMT_MTD.3***, *FMT_SMF.1 and FMT_SMR.1*) and is only accessible to user-applications through the API (***FPT_RVM.1***).

O.FIREWALL

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

O.NATIVE

The *Java Card VM* is the machine running the bytecode of the *applet*s (***FPT_RVM.1***). These can only be linked with API methods or other *package*s already on the card. This objective mainly relies on the environmental objective *OE.NATIVE*, which upholds the assumption *A.NATIVE*.

O.REALLOCATION

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

O.SHRD_VAR_CONFID

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

O.SHRD_VAR_INTEG

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

SERVICES

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

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

Transactions are provided to *applet*s as Java Card technology-based class libraries ("Java Card class libraries").

O.CIPHER This objective is directly related to ***FCS_CKM.1***, ***FCS_CKM.2***, ***FCS_CKM.3***, ***FCS_CKM.4*** and ***FCS_COP.1***. Another important SFR is ***FPR_UNO.1***, the observation of the cryptographic operations may be used to disclose the keys.

The associated security functions are not described herein. They are provided to *applet*s as Java Card class libraries (see the class **`javacardx.crypto.Cipher`** and the package **`javacardx.security`**).

O.PIN-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements. API classes implement the security functions behind these requirements. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the objects.

O.KEY-MNGT This relies on the same functional requirements as *O.CIPHER*, plus *FDP_RIP.1* and ***FDP_SDI.2*** as well.

	FAU_ARP.1	FCS_CKM.1	FCS_CKM.2	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FDP_ROL.1	FDP_SDI.2	FIA_ATD.1	FIA_UID.2	FIA_USB.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_MTD.3	FMT_SMR.1	FMT_SMF.1	FPR_UNO.1	FPT_FLS.1	FPT_RVM.1	FPT_SEP.1	FPT_TDC.1	FPT_TST.1
O.ALARM	X																								X				
O.CIPHER		X	X	X	X	X																		X					
O.FIREWALL							X	X	X	X							X	X	X	X	X	X	X			X	X		
O.KEY-MNGT		X	X	X	X	X					X		X											X					
O.NATIVE																										X			
O.OPERATE	X						X	X						X		X									X	X	X	X	X
O.PIN-MNGT							X	X			X	X	X											X					
O.RESOURCES	X																			X	X	X			X	X			
O.SID														X	X	X	X		X	X	X		X			X	X		
O.TRANSACTION											X	X																	
O.SHRD_VAR_CONFID									X	X	X																		
O.SHRD_VAR_INTEG									X	X																			
O.REALLOCATION											X																		

Table 11: Security requirements rationale for the Minimal Configuration

6.2.1.2 IT Environment Security Requirements Rationale

The environmental objective *OE.1224HVERIFICATION*, which is satisfied by IT procedural means, is met by the SFRs of the group *BCVG* (§5.1.3).

The environmental objective *OE.CARD-MANAGEMENT*, which is satisfied by IT procedural means, is met by the SFRs of the group *CMGRG* (§5.1.10).

All the security functional requirements to which this section makes reference from now on are those specified in the group *SCPG* (§5.1.9).

The components *FPT_RCV.3* and *FPT_RCV.4* are used to support the objective *OE.1212H*SCP.*SUPPORT* and *OE.1210H*SCP.*RECOVERY* to assist the TOE to recover in the event of a power failure. If the power fails or the card is withdrawn prematurely from the CAD, the operation of the TOE may be interrupted leaving the TOE in an inconsistent state.

OE.SCP.RECOVERY This objective is met by the components *FPT_FLS.1*, *FPT_RCV.3* and *FRU_FLT.1*.

OE.SCP.SUPPORT This objective is met by the components *FPT_SEP.1* (no bypassing TSF), *FPT_AMT.1*, *FPT_RCV.3*, *FPT_RCV.4* and *FPT_RVM.1*.

OE.SCP.IC This objective is met by the component *FPT_PHP.3*.

	FPT_AMT.1	*FPT_FLS.1*	*FPT_PHP.3*	*FPT_RCV.3*	*FPT_RCV.4*	*FPT_RVM.1*	*FPT_SEP.1*	*FRU_FLT.1*
OE.SCP.RECOVERY		X		X				X
OE.SCP.SUPPORT	X			X	X	X	X	
OE.SCP.IC			X					

Table 12: Security requirements rationale for the group *SCPG*

6.2.1.3 Security Functional Requirements Dependencies

The TOE assurance requirements dependencies for level EAL4 are completely fulfilled.

The functional requirements dependencies for the TOE are not completely fulfilled. The **KO**s in the following table corresponds to unsatisfied dependencies that are explained and justified in the rationale that appears right below the table.

SFR	Dependency	Status
FAU_ARP.1/JCS	(FAU_SAA.1)	KO: FAU_SAA.1 is not satisfied
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.2, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.2	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.3	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.4	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FMT_MSA.2)	OK: FCS_CKM.1, FMT_MSA.2/JCRE
FCS_COP.1	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FDP_ACC.1/CMGR	(FDP_ACF.1)	OK: FDP_ACF.1/CMGR
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	OK: FDP_ACF.1/FIREWALL
FDP_ACF.1/CMGR	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/CMGR, FMT_MSA.3/CMGR
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL
FDP_IFC.1/JCVM	(FDP_IFF.1)	OK: FDP_IFF.1/JCVM
FDP_IFC.2/BCV	(FDP_IFF.1)	OK: FDP_IFF.2/BCV
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCVM, FMT_MSA.3/FIREWALL
FDP_IFF.2/BCV	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/BCV, FMT_MSA.3/BCV
FDP_RIP.1	None	OK
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM
FDP_SDI.2	None	OK
FIA_ATD.1/AID	None	OK
FIA_UID.1/CMGR	None	OK
	None	OK
FIA_USB.1	(FIA_ATD.1)	OK: FIA_ATD.1/AID
FMT_MSA.1/BCV	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/BCV, FMT_SMR.1/BCV, ***FMT_SMF.1.1/BCV***

SFR	Dependency	Status
FMT_MSA.1/CMGR	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.1/CMGR, FMT_SMR.1/CMGR, *FMT_SMF.1.1/CMGR*
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MSA.2/BCV	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_IFC.2/BCV, FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.2/JCRE	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.3/BCV	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/CMGR	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CMGR, FMT_SMR.1/CMGR
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MTD.1/JCRE	(FMT_SMR.1) and (FMT_SMF.1)	OK: FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MTD.3	(ADV_SPM.1) and (FMT_MTD.1)	OK: FMT_MTD.1/JCRE
FMT_SMR.1/BCV	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/CMGR	(FIA_UID.1)	OK: FIA_UID.1/CMGR
FMT_SMR.1/JCRE	(FIA_UID.1)	OK:
FPR_UNO.1	None	OK
FPT_AMT.1/SCP	None	OK
FPT_FLS.1/JCS	(ADV_SPM.1)	OK
FPT_FLS.1/SCP	(ADV_SPM.1)	OK
FPT.PHP.3/SCP	None	OK
FPT_RCV.3/SCP	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK:FPT_TST.1
FPT_RCV.4/SCP	(ADV_SPM.1)	OK
FPT_RVM.1	None	OK
FPT_RVM.1/SCP	None	OK
FPT_SEP.1	None	OK
FPT_SEP.1/SCP	None	OK
FPT_TDC.1	None	OK
FPT_TST.1	(FPT_AMT.1)	OK: FPT_AMT.1/SCP
FRU_RSA.1/BCV	None	OK
FRU_FLT.1/SCP	(FPT_FLS.1)	OK: FPT_FLS.1/SCP

Table 13: Functional Requirement Dependencies (Minimal)

FAU_SAA.1 Potential violation analysis is used to specify the set of auditable events whose occurrence or accumulated occurrence held to indicate a potential violation of the TSP, and any rules to be used to perform the violation analysis. The dependency of FAU_ARP.1/JCS on this functional requirement assumes that a "potential security violation" is an audit event indicated by the FAU_SAA.1 component. The events listed in FAU_ARP.1/JCS are, on the contrary, merely self-contained ones (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The *Java Card VM* or other components of the TOE detect these events during their usual working order. Thus, in principle there would be no applicable audit recording in this framework. Moreover, no specification of one such recording is provided elsewhere. Therefore no set of auditable events could possibly be defined.

FIA_UID.1 This is required by the component ***FMT_SMR.1*** of the group *BCVG*. However, the role bytecode verifier defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The bytecode verifier does not "identify" itself with respect to the TOE, furthermore, it is part of the IT environment. Thus, here it is claimed that this dependency can be left out.

6.2.1.4 Rationale for Strength of Function Medium

The minimum strength of function level required is **SOF-medium**.

The TOE is intended to operate in open environments, where attackers can easily exploit vulnerabilities. According to the claimed intended usage of the TOE, it is very likely that it may represent a significant value and then constitute an attractive target for attacks. In some malicious usages of the TOE the statistical or probabilistic mechanisms in the TOE, for instance, may be subjected to analysis and attack in the normal course of operation. A medium strength of function level to be the reasonable minimum level for cards hosting sensitive applications. It shall probably be the case, as it is frequent nowadays, that the required strength of function level will be high in, for instance, banking or electronic signature applications. Considering that Java Card technology-based products may also address other less security sensitive contexts, and furthermore, that the resistance of the mechanisms mentioned above to attacks with high potential is hard to be achieved and demonstrated, the choice of a high strength of function requirement is left to the card issuer depending on the intended usage of the product. Thus, in this protection profile, a protection against moderate attack potential has been chosen as the minimal level for those multi-applicative cards.

The strength of function level medium is consistent with the vulnerability analysis level that has been specified (***AVA_VLA.3).***

6.2.1.5 Rationale for Assurance Level EAL4 augmented

The assurance level for this protection profile is EAL4 augmented. Augmentation results from the selection of the components ***AVA_VLA.3*** and ***ADV_IMP.2.***

6.2.1.5.1 Rationale for Assurance Level EAL4

EAL4 allows a developer to attain a reasonably high assurance level without the need for highly specialized processes and practices. It corresponds to a white box analysis and it can be considered as a reasonable level that can be applied to an existing product line without undue expense and complexity.

6.2.1.5.2 Rationale for Augmentation

The evaluation of the TOE may be performed, for instance, because the product hosts one or several sensitive applications, such as financial and health recording ones, which contain, represent, or provide access to valuable assets. In addition to that the TOE may not be directly under the control of trained and dedicated administrators.

AVA_VLA.3

As a result, it is imperative that the TOE vulnerabilities to be reviewed be drawn from a systematic search rather than strictly a manufacturer prepared identification list. Component ***AVA_VLA.3*** requires that such a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***. There might be scenarios, for example if the TOE is intended to stay in a hostile environment for long periods of time, or if the applications are considered to be highly sensitive, that would justify a further augmentation by requiring the component ***AVA_VLA.4.*** This latter component dictates that the TOE must be shown to be resistant to penetration attacks performed by attackers possessing a high attack potential. The choice of augmenting the assurance level using the component ***AVA_VLA.4*** is left to the card issuer.

AVA_VLA.3 has the following dependencies:

- ***ADV_FSP.1*** Informal functional specification
- ***ADV_HLD.2*** Security enforcing high-level design
- ***ADV_IMP.1*** Subset of the implementation of the TSF
- ***ADV_LLD.1*** Descriptive low-level design
- ***AGD_ADM.1*** Administrator guidance
- ***AGD_USR.1*** User guidance

All of these are met or exceeded in the EAL4 assurance package.

ADV_IMP.2

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement.

The assurance component ***ADV_IMP.2*** has been chosen because the evaluation of the TOE must ensure that its security functional requirements are completely and accurately addressed by the implementation representation of the TSF.

ADV_IMP.2 has the following dependencies:

- ***ADV_LLD.1*** Descriptive low-level design
- ***ADV_RCR.1*** Informal correspondence demonstration
- ***ALC_TAT.1*** Well-defined development tools

All of these are met or exceeded in the EAL4 assurance package.

6.2.1.6 Internal Consistency and Mutual Support

The purpose of this part of the rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given to this because:

The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies are satisfied).

The dependencies analysis for the functional requirements described in the section "Security Functional Requirements Dependencies" demonstrates mutual support and internal consistency between the functional requirements. That analysis also shows that the dependencies between functional and assurance requirements are also satisfied.

6.2.2 Java Card System Standard 2.1.1 Configuration

6.2.2.1 TOE Security Requirements Rationale

Unless explicitly stated, all the security functional requirements to which this section makes reference are those specified in the groups CoreG (§5.1.1), InstG (§5.1.2) and CarG (§5.1.8).

Note: the differences between the **Minimal** and the **Java Card System Standard 2.1.1** configurations have been underlined in the following rationale.

IDENTIFICATION

O.SID Subjects' identity is *AID*-based (*applet*s, *package*s), and is met by *FDP_ITC.2*, *FIA_ATD.1*, ***FMT_MSA.1***, ***FMT_MSA.3***, *FMT_SMF.1*, ***FMT_MTD.1***, and ***FMT_MTD.3***. Additional support includes ***FPT_RVM.1*** and ***FPT_SEP.1***.

At last, installation procedures ensure protection against forgery (the *AID* of an applet is under the control of the TSFs) or re-use of identities (***FIA_UID.2***, ***FIA_USB.1***).

APPLET MANAGEMENT

O.INSTALL This objective specifies that installation of *applets* must be secure. Security attributes of installed data are under the control of the *FIREWALL* access control *policy* (*FDP_ITC.2*), and the TSFs are protected against possible failures of the *installer* (*FPT_FLS.1/Installer, FPT_RCV.3*).

O.LOAD This objective specifies that the loading of a *package* into the card must be secure. Evidence of the origin of the package is enforced (*FCO_NRO.2*) and the integrity of the corresponding data is under the control of the ***PACKAGE LOADING information flow*** policy (*FDP_IFC.2/CM, FDP_IFF.1/CM)* and *FDP_UIT.1.* Appropriate identification (*FIA_UID.1/CM*) and transmission mechanisms are also enforced (FTP_ITC.1).

EXECUTION

O.OPERATE The TOE is protected in various ways against *applet*s' actions (***FPT_RVM.1***, ***FPT_SEP.1***, *FPT_TDC.1*), the ***FIREWALL*** *access control policy* (*FDP_ACC.2*, *FDP_ACF.1*), and is able to detect and block various failures or security violations during usual working (*FPT_FLS.1*, *FAU_ARP.1*). Startup of the TOE is covered by *FPT_TST.1*, and indirectly by *FPT_AMT.1* (this latter defined in group *SCPG* §5.1.9). .

Its security-critical parts and procedures are also protected: safe recovery from failure is ensured (*FPT_RCV.3*), *applets'* installation may be cleanly

aborted (*FDP_ROL.1*), communication with external users and their internal subjects is well-controlled (*FDP_ITC.2*, *FIA_ATD.1*, ***FIA_USB.1***) to prevent alteration of TSF data (also protected by components of the FPT class).

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

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

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

O.NATIVE The *Java Card VM* is the machine running the bytecode of the *applet*s (***FPT_RVM.1***). These can only be linked with API methods or other *package*s already on the card. This objective mainly relies on the environmental objectives *OE.NATIVE* and *OE.APPLET*, which uphold the assumptions *A.1023HNATIVE* and *A.APPLET* respectively.

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

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

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

SERVICES

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

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

Transactions are provided to *applet*s as Java Card class libraries.

O.CIPHER This objective is directly related to ***FCS_CKM.1***, ***FCS_CKM.2***, ***FCS_CKM.3***, ***FCS_CKM.4*** and ***FCS_COP.1***. Another important SFR is ***FPR_UNO.1***, the observation of the cryptographic operations may be used to disclose the keys.

The associated security functions are not described herein. They are provided to *applet*s as Java Card class libraries, (see the class **`javacardx.crypto.Cipher`** and the package **`javacardx.security`**).

O.PIN-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements.API classes implement the security functions behind these requirements. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the objects.

O.KEY-MNGT This relies on the same functional requirements as *O.CIPHER*, plus *FDP_RIP.1* and ***FDP_SDI.2*** as well.

	FAU_ARP.1	FCS_CKM.1	FCS_CKM.2	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FDP_ROL.1	FDP_SDI.2	FIA_ATD.1	FIA_UID.2	FIA_USB.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_MTD.3	FMT_SMR.1	FMT_SMF.1	FPR_UNO.1	FPT_FLS.1	FPT_RVM.1	FPT_SEP.1	FPT_TDC.1	FPT_TST.1
O.ALARM	X																								X				
O.CIPHER		X	X	X	X	X																		X					
O.FIREWALL							X	X	X	X							X	X	X	X	X	X	X			X	X		
O.KEY-MNGT		X	X	X	X	X					X		X											X					
O.NATIVE																										X			
O.OPERATE	X						X	X				X		X		X									X	X	X	X	X
O.PIN-MNGT							X	X			X	X	X											X					
O.RESOURCES	X											X								X	X	X			X	X			
O.SID														X	X	X	X		X	X	X		X			X	X		
O.TRANSACTION											X	X																	
O.SHRD_VAR_CONFID									X	X	X																		
O.SHRD_VAR_INTEG									X	X																			
O.REALLOCATION											X																		

	FCO_NRO.2	FDP_IFC.2	FDP_IFF.1	FDP_ITC.2	FDP_UIT.1	FIA_UID.1	FPT_FLS.1	FPT_RCV.3	FRU_RSA.1	FTP_ITC.1
O.INSTALL				X			X	X		
O.LOAD	X	X	X		X	X				X
O.SID				X						
O.OPERATE				X				X		
O.RESOURCES								X	X	
O.FIREWALL				X						

Table 14: Security requirements rationale for the Java Card System Standard 2.1.1 **Configuration**

6.2.2.2 IT Environment Security Requirements Rationale

The environmental objective *OE.1224HVERIFICATION*, which is satisfied by IT procedural means, is met by the SFRs of the group *BCVG* (§5.1.3).

The environmental objective *OE.APPLET* might be also satisfied by IT procedural means. The IT verification that a post-issuance loaded applet contains no native code could be carried out as a part of the verification of how well the *CAP* file is formed. This verification has been associated in the group *BCVG* (§5.1.3) to the requirement of secure security attributes, expressed by the component ***FMT_MSA.2*** (see application note at pp. 90).

The environmental objective *OE.CARD-MANAGEMENT*, which is satisfied by IT procedural means, is met by the SFRs of the group *CMGRG* (§5.1.10).

All the security functional requirements to which this section makes reference from now on are those specified in the group *SCPG* (§5.1.9).

The components *FPT_RCV.3* and *FPT_RCV.4* are used to support the objective *OE.1212H*SCP.*SUPPORT* and *OE.1210H*SCP.*RECOVERY* to assist the TOE to recover in the event of a power failure. If the power fails or the card is withdrawn prematurely from the CAD the operation of the TOE may be interrupted leaving the TOE in an inconsistent state.

OE.SCP.RECOVERY This objective is met by the components *FPT_FLS.1*, *FPT_RCV.3* and *FRU_FLT.1*.

OE.SCP.SUPPORT This objective is met by the components *FPT_SEP.1* (no bypassing TSF), *FPT_AMT.1*, *FPT_RCV.3*, *FPT_RCV.4* and *FPT_RVM.1*.

OE.SCP.IC This objective is met by the component *FPT_PHP.3*.

	FPT_AMT.1	*FPT_FLS.1*	*FPT_PHP.3*	*FPT_RCV.3*	*FPT_RCV.4*	*FPT_RVM.1*	*FPT_SEP.1*	*FRU_FLT.1*
OE.SCP.RECOVERY		X		X				X
OE.SCP.SUPPORT	X			X	X	X	X	
OE.SCP.IC			X					

Table 15: Security requirements rationale for the group *SCPG*

6.2.2.3 Security Functional Requirements Dependencies

The TOE assurance requirements dependencies for level EAL4 are completely fulfilled.

The functional requirements dependencies for the TOE are not completely fulfilled. The KOs in the following table corresponds to unsatisfied dependencies that are explained and justified in the rationale that appears below the table.

SFR	Dependency	Status
FAU_ARP.1/JCS	(FAU_SAA.1)	KO: FAU_SAA.1 is not satisfied
FCO_NRO.2	(FIA_UID.1)	OK: FIA_UID.1/CM
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.2, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.2	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.3	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.4	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FMT_MSA.2)	OK: FCS_CKM.1, FMT_MSA.2/JCRE
FCS_COP.1	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FDP_ACC.1/CMGR	(FDP_ACF.1)	OK: FDP_ACF.1/CMGR
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	OK: FDP_ACF.1/FIREWALL
FDP_ACF.1/CMGR	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/CMGR, FMT_MSA.3/CMGR
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL
FDP_IFC.1/JCVM	(FDP_IFF.1)	OK: FDP_IFF.1/JCVM
FDP_IFC.2/BCV	(FDP_IFF.1)	OK: FDP_IFF.2/BCV
FDP_IFC.2/CM	(FDP_IFF.1)	OK: FDP_IFF.1/CM
FDP_IFF.1/CM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/CM, FMT_MSA.3/CM
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCVM, FMT_MSA.3/FIREWALL
FDP_IFF.2/BCV	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/BCV, FMT_MSA.3/BCV
FDP_ITC.2/Installer	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1) and (FPT_TDC.1)	OK: FPT_TDC.1, FDP_IFC.2/CM, FTP_ITC.1/CM

SFR	Dependency	Status
FDP_RIP.1	None	OK
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM
FDP_SDI.2	None	OK
FDP_UIT.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1)	OK: FDP_IFC.2/CM, FTP_ITC.1/CM
FIA_ATD.1/AID	None	OK
FIA_UID.1/CM	None	OK
FIA_UID.1/CMGR	None	OK
	None	OK
FIA_USB.1	(FIA_ATD.1)	OK: FIA_ATD.1/AID
FMT_MSA.1/BCV	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/BCV, FMT_SMR.1/BCV, ***FMT_SMF.1.1/BCV***
FMT_MSA.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/CM, FMT_SMR.1/CM, *FMT_SMF.1.1/CM*
FMT_MSA.1/CMGR	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.1/CMGR, FMT_SMR.1/CMGR, *FMT_SMF.1.1/CMGR*
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MSA.2/BCV	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_IFC.2/BCV, FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.2/JCRE	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.3/BCV	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/CM	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CM, FMT_SMR.1/CM
FMT_MSA.3/CMGR	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CMGR, FMT_SMR.1/CMGR
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRE, FMT_SMR.1/JCRE

SFR	Dependency	Status
FMT_MTD.1/JCRE	(FMT_SMR.1) and (FMT_SMF.1)	OK: FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MTD.3	(ADV_SPM.1) and (FMT_MTD.1)	OK: FMT_MTD.1/JCRE
FMT_SMR.1/BCV	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/CM	(FIA_UID.1)	OK: FIA_UID.1/CM
FMT_SMR.1/CMGR	(FIA_UID.1)	OK: FIA_UID.1/CMGR
FMT_SMR.1/JCRE	(FIA_UID.1)	OK:***FIA_UID.2.1/AID***
FMT_SMR.1/Installer	(FIA_UID.1)	KO: FIA_UID.1
FPR_UNO.1	None	OK
FPT.PHP.3/SCP	None	OK
FPT_AMT.1/SCP	None	OK
FPT_FLS.1/Installer	(ADV_SPM.1)	OK
FPT_FLS.1/JCS	(ADV_SPM.1)	OK
FPT_FLS.1/SCP	(ADV_SPM.1)	OK
FPT_RCV.3/Installer	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK: FPT_TST.1
FPT_RCV.3/SCP	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK: FPT_TST.1
FPT_RCV.4/SCP	(ADV_SPM.1)	OK
FPT_RVM.1	None	OK
FPT_RVM.1/SCP	None	OK
FPT_SEP.1	None	OK
FPT_SEP.1/SCP	None	OK
FPT_TDC.1	None	OK
FPT_TST.1	(FPT_AMT.1)	OK: FPT_AMT.1/SCP
FRU_FLT.1/SCP	(FPT_FLS.1)	OK: FPT_FLS.1/SCP
FRU_RSA.1/Installer	None	OK
FRU_RSA.1/BCV	None	OK
FTP_ITC.1/CM	None	OK

Table 16: Functional Requirement Dependencies (Java Card System Standard 2.1.1**)**

FAU_SAA.1

Potential violation analysis is used to specify the set of auditable events whose occurrence or accumulated occurrence held to indicate a potential violation of the TSP, and any rules to be used to perform the violation analysis. The dependency of FAU_ARP.1/JCS on this functional requirement assumes that a "potential security violation" is an audit event indicated by the FAU_SAA.1 component. The events listed in FAU_ARP.1/JCS are, on the contrary, merely self-contained ones (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The *Java Card VM* or other components of the TOE detect these events during their usual working order. Thus, in principle there would be no applicable audit recording in this framework. Moreover, no specification of one such recording is provided elsewhere. Therefore no set of auditable events could possibly be defined.

FIA_UID.1 This is required by the component ***FMT_SMR.1*** of the group *InstG*. However, the role installer defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The installer does not "identify" itself with respect to the TOE, but is a part of it. Thus, here it is claimed that this dependency can be left out. The reader may notice that the role is required because of the SFRs on management of TSF data and security attributes, essentially those of the firewall policy.

This is also required by the component ***FMT_SMR.1*** of the group *BCVG*. However, the role bytecode verifier defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The bytecode verifier does not "identify" itself with respect to the TOE; furthermore, it is part of the IT environment. Thus, here it is claimed that this dependency can be left out.

6.2.2.4 Rationale for Strength of Function Medium

The minimum strength of function level required is **SOF-medium**.

The TOE is intended to operate in open environments, where attackers can easily exploit vulnerabilities. According to the claimed intended usage of the TOE, it is very likely that it may represent a significant value and then constitute an attractive target for attacks. In some malicious usages of the TOE the statistical or probabilistic mechanisms in the TOE, for instance, may be subjected to analysis and attack in the normal course of operation. A medium strength of function seems to be the reasonable minimum level for cards hosting sensitive applications. It shall probably be the case, as it is frequent nowadays, that the required strength of function level will be high in, for instance, banking or electronic signature applications. Considering that Java Card technology-based products may also address other less security sensitive contexts, and furthermore, that the resistance of the mechanisms mentioned above to attacks with high potential is hard to be achieved and demonstrated, the choice of a high strength of function requirement is left to the card issuer depending on the intended usage of the product. Thus, in this protection profile, a protection against moderate attack potential has been chosen as the minimal level for those multi-applicative cards.

The strength of function level medium is consistent with the vulnerability analysis level that has been specified (***AVA_VLA.3).***

6.2.2.5 Rationale for Assurance Level EAL4 augmented

The assurance level for this protection profile is EAL4 augmented. Augmentation results from the selection of the components ***AVA_VLA.3*** and ***ADV_IMP.2.***

6.2.2.5.1 Rationale for Assurance Level EAL4

EAL4 allows a developer to attain a reasonably high assurance level without the need for highly specialized processes and practices. It corresponds to a white box analysis and it can be considered as a reasonable level that can be applied to an existing product line without undue expense and complexity.

6.2.2.5.2 Rationale for Augmentation

The evaluation of the TOE may be performed, for instance, because the product hosts one or several sensitive applications, such as financial and health recording ones, which contain, represent, or provide access to valuable assets. In addition to that the TOE may not be directly under the control of trained and dedicated administrators.

AVA_VLA.3

As a result, it is imperative that the TOE vulnerabilities to be reviewed be drawn from a systematic search rather than strictly a manufacturer prepared identification list. Component ***AVA_VLA.3*** requires that such a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***. There might be scenarios, for example if the TOE is intended to stay in a hostile environment for long periods of time, or if the applications are considered to be highly sensitive, that would justify a further augmentation by requiring the component ***AVA_VLA.4.*** This latter component dictates that the TOE must be shown to be resistant to penetration attacks performed by attackers possessing a high attack potential. The choice of augmenting the assurance level using the component ***AVA_VLA.4*** is left to the card issuer.

AVA_VLA.3 has the following dependencies:

ADV_FSP.1 Informal functional specification

ADV_HLD.2 Security enforcing high-level design

ADV_IMP.1 Subset of the implementation of the TSF

ADV_LLD.1 Descriptive low-level design

AGD_ADM.1 Administrator guidance

AGD_USR.1 User guidance

All of these are met or exceeded in the EAL4 assurance package.

ADV_IMP.2

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement.

The assurance component ***ADV_IMP.2*** has been chosen because the evaluation of the TOE must ensure that its security functional requirements are completely and accurately addressed by the implementation representation of the TSF.

ADV_IMP.2 has the following dependencies:

ADV_LLD.1 Descriptive low-level design

ADV_RCR.1 Informal correspondence demonstration

ALC_TAT.1 Well-defined development tools

All of these are met or exceeded in the EAL4 assurance package.

6.2.2.6 *Internal Consistency and Mutual Support*

The purpose of this part of the Protection Profile rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given to this because:

The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies are satisfied).

The dependencies analysis for the functional requirements described in the section "Security Functional Requirements Dependencies" demonstrates mutual support and internal consistency between the functional requirements. That analysis also shows that the dependencies between functional and assurance requirements are also satisfied.

6.2.3 Java Card System Standard 2.2 Configuration

6.2.3.1 TOE Security Requirements Rationale

In the context of this rationale the *FIREWALL access control policy* is the one specified in the group LCG (§5.1.6). The references to the components *FDP_ACC.2/FIREWALL*, *FDP_ACF.1/FIREWALL* and *FMT_MSA.1/JCRE* must be understood as denoting the definitions of those components as provided in the group LCG.

Note: The differences between the **Java Card System Standard 2.1.1** and the **Java Card System Standard 2.2** configurations have been underlined in the following rationale.

IDENTIFICATION

O.SID Subjects' identity is *AID*-based (*applet*s, *package*s), and is met by *FDP_ITC.2*, *FIA_ATD.1*, ***FMT_MSA.1***, ***FMT_MSA.3***, *FMT_SMF.1*, ***FMT_MTD.1***, and ***FMT_MTD.3***. Additional support includes ***FPT_RVM.1*** and ***FPT_SEP.1***.

Lastly, installation procedures ensure protection against forgery (the *AID* of an applet is under the control of the TSFs) or re-use of identities (***FIA_UID.2***, ***FIA_USB.1***).

APPLET MANAGEMENT

O.INSTALL This objective specifies that installation of *applets* must be secure. Security attributes of installed data are under the control of the *FIREWALL* access control *policy* (*FDP_ITC.2*), and the TSFs are protected against possible failures of the *installer* (*FPT_FLS.1/Installer*, *FPT_RCV.3*).

O.LOAD This objective specifies that the loading of a *package* into the card must be secure. Evidence of the origin of the package is enforced (*FCO_NRO.2*) and the integrity of the corresponding data is under the control of the ***PACKAGE LOADING information flow*** policy (*FDP_IFC.2/CM*, *FDP_IFF.1/CM)* and *FDP_UIT.1*. Appropriate identification (*FIA_UID.1/CM*) and transmission mechanisms are also enforced *(FTP_ITC.1)*.

O.1186HDELETION This objective specifies that *applet* and *package* deletion must be secure. The non-introduction of security holes is ensured by the *ADEL access control* ***policy*** (*FDP_ACC.2/ADEL, FDP_ACF.1/ADEL*). The integrity and confidentiality of data that does not belong to the deleted *applet* or *package* is a by-product of this policy as well. Non-accessibility of deleted data is met by *FDP_RIP.1/ADEL* and the TSFs are protected against possible failures of the deletion procedures (*FPT_FLS.1/ADEL*, *FPT_RCV.3* (see application note)). The functional requirements of the class ***FMT*** included in the group *ADELG* also contribute to meet this objective.

EXECUTION

O.OPERATE The TOE is protected in various ways against *applet*s' actions (***FPT_RVM.1***, ***FPT_SEP.1***, *FPT_TDC.1*), the ***FIREWALL*** *access control policy* (*FDP_ACC.2*, *FDP_ACF.1*), and is able to detect and block various failures or security violations during usual working (*FPT_FLS.1*, *FAU_ARP.1*). Startup of the TOE is covered by *FPT_TST.1*, and indirectly by *FPT_AMT.1* (this latter defined in group *SCPG* §5.1.9).

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

Almost every objective and/or functional requirement indirectly contributes to this one too, in particular the EXTERNAL MEMORY access control policy (*FDP_ACC.1*, *FDP_ACF.1*), if the TOE implements external memory features (Java Card platform, version 2.2.2).

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

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

O.NATIVE The *Java Card VM* is the machine running the bytecode of the *applet*s (***FPT_RVM.1***). These can only be linked with API methods or other *package*s already on the card. This objective mainly relies on the environmental objectives ***OE.NATIVE*** and *OE.APPLET*, which uphold the assumptions *A.1023HNATIVE* and *A.APPLET* respectively.

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

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

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

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

SERVICES

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

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

Transactions are provided to *applet*s as Java Card class libraries.

O.CIPHER This objective is directly related to ***FCS_CKM.1***, ***FCS_CKM.2***, ***FCS_CKM.3***, ***FCS_CKM.4*** and ***FCS_COP.1***. Another important SFR is ***FPR_UNO.1***, the observation of the cryptographic operations may be used to disclose the keys.

The associated security functions are not described herein. They are provided to *applet*s as Java Card class libraries (see the class **`javacardx.crypto.Cipher`** and the package **`javacard.security`**).

O.PIN-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements.API classes implement the security functions behind these. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the objects.

O.BIO-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements. The applets that manage biometric templates rely on the security functions that implement these requirements. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the templates. Note that the objective applies only to configurations including the javacardx.biometry package defined in [JCAPI222].

O.KEY-MNGT This relies on the same functional requirements as *O.CIPHER*, plus *FDP_RIP.1* and ***FDP_SDI.2*** as well.

O.REMOTE The access to the TOE's internal data and the flow of information from the card to the *CAD* required by the *Java Card RMI* service is under control of the *Java Card RMI access control policy* (*FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI*) and the ***JCRMI*** *information flow* control policy (*FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI*). The functional requirements of the class ***FMT*** included in the group *RMIG* also contribute to meet this objective.

O.EXT_MEM The Java Card System memory is protected against applet's tentatives of unauthorized access through the external memory facilities by the *EXTERNAL MEMORY access control policy* (***FDP_ACC.1.1/EXT_MEM, FDP_ACF.1.1/ EXT_MEM***), which first controls the accessible address space, then controls the effective read and write operations.

OBJECT DELETION

O.1189HOBJ-DELETION This objective specifies that deletion of objects is secure. The objective is met by the functional requirements *FDP_RIP.1/ODEL* and *FPT_FLS.1/ODEL.*

	FAU_ARP.1	FCS_CKM.1	FCS_CKM.2	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.1	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FDP_ROL.1	FDP_SDI.2	FIA_ATD.1	FIA_UID.2	FIA_USB.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_MTD.3	FMT_SMR.1	FMT_SMF.1	FPR_UNO.1	FPT_FLS.1	FPT_RVM.1	FPT_SEP.1	FPT_TDC.1	FPT_TST.1
O.ALARM	X																									X				
O.CIPHER		X	X	X	X	X																			X					
O.FIREWALL								X	X	X	X							X	X	X	X	X	X	X			X	X		
O.KEY-MNGT		X	X	X	X	X						X		X											X					
O.NATIVE																											X			
O.OPERATE	X							X	X				X		X		X									X	X	X	X	X
O.PIN-MNGT								X	X			X	X	X											X					
O.BIO-MNGT								X	X			X	X	X																
O.RESOURCES	X												X								X	X	X			X	X			
O.SID															X	X	X	X		X	X	X		X			X	X		
O.TRANSACTION												X	X																	
O.SHRD_VAR_CONFID										X	X	X																		
O.SHRD_VAR_INTEG										X	X																			
O.REALLOCATION												X																		
O.EXT_MEM							X		X									X	X	X			X	X						

	FCO_NRO.2	FDP_IFC.2	FDP_IFF.1	FDP_ITC.2	FDP_UIT.1	FIA_UID.1	FPT_FLS.1	FPT_RCV.3	FRU_RSA.1	FTP_ITC.1
O.INSTALL				X			X	X		
O.LOAD	X	X	X		X	X				X
O.SID				X						
O.OPERATE				X				X		
O.RESOURCES								X	X	
O.FIREWALL				X						

	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FMT_MSA.1	FMT_MSA.3	FMT_REV.1	FMT_SMR.1	FPT_FLS.1	FPT_RCV.3
O.DELETION	X	X			X	X	X		X	X	X
O.OBJ-DELETION					X					X	
O.REMOTE	X	X	X	X		X	X	X	X		
O.FIREWALL								X			

Table 17: Security requirements rationale for the Java Card System Standard 2.2 **Configuration**

6.2.3.2 IT Environment Security Requirements Rationale

The environmental objective *OE.1224HVERIFICATION*, which is satisfied by IT procedural means, is met by the SFRs of the group *BCVG* (§5.1.3).

The environmental objective *OE.APPLET* might be also satisfied by IT procedural means. The IT verification that a post-issuance loaded applet contains no native code could be carried out as a part of the verification of how well the *CAP* file is formed. This verification has been associated in the group *BCVG* (§5.1.3) to the requirement of secure security attributes, expressed by the component ***FMT_MSA.2*** (see application note at pp. 90).

The environmental objective *OE.CARD-MANAGEMENT*, which is satisfied by IT procedural means, is met by the SFRs of the group *CMGRG* (§5.1.10).

All the security functional requirements to which this section makes reference from now on are those specified in the group *SCPG* (§5.1.9).

The components *FPT_RCV.3* and *FPT_RCV.4* are used to support the objective *OE.1212H*SCP.*SUPPORT* and *OE.1210H*SCP.*RECOVERY* to assist the TOE to recover in the event of a power failure or signal loss. If the RF signal is lost, the power fails or the card is withdrawn prematurely from the CAD, the operation of the TOE may be interrupted leaving the TOE in an inconsistent state.

OE.SCP.RECOVERY This objective is met by the components *FPT_FLS.1*, *FPT_RCV.3* and *FRU_FLT.1*.

OE.SCP.SUPPORT This objective is met by the components *FPT_SEP.1* (no bypassing TSF), *FPT_AMT.1*, *FPT_RCV.3*, *FPT_RCV.4* and *FPT_RVM.1*.

OE.SCP.IC This objective is met by the component *FPT_PHP.3*.

	FPT_AMT.1	FPT_FLS.1	FPT_PHP.3	FPT_RCV.3	FPT_RCV.4	FPT_RVM.1	FPT_SEP.1	FRU_FLT.1
OE.SCP.RECOVERY		X		X				X
OE.SCP.SUPPORT	X			X	X	X	X	
OE.SCP.IC			X					

Table 18: Security requirements rationale for the group *SCPG*

6.2.3.3 Security Functional Requirements Dependencies

The TOE assurance requirements dependencies for level EAL4 are completely fulfilled.

The functional requirements dependencies for the TOE are not completely fulfilled. The KOs in the following table corresponds to unsatisfied dependencies that are explained and justified in the rationale that appears below the table.

SFR	Dependency	Status
FAU_ARP.1/JCS	(FAU_SAA.1)	KO: FAU_SAA.1 is not satisfied
FCO_NRO.2/CM	(FIA_UID.1)	OK: FIA_UID.1/CM
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.2, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.2	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.3	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.4	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FMT_MSA.2)	OK: FCS_CKM.1, FMT_MSA.2/JCRE
FCS_COP.1	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FDP_ACC.1/CMGR	(FDP_ACF.1)	OK: FDP_ACF.1/CMGR
FDP_ACC.2/ADEL	(FDP_ACF.1)	OK:FDP_ACF.1/ADEL
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	OK: FDP_ACF.1/FIREWALL

SFR	Dependency	Status
FDP_ACC.1.1/EXT_MEM	(FDP_ACF.1)	OK: ***FDP_ACF.1.1/ EXT_MEM***
FDP_ACC.2/JCRMI	(FDP_ACF.1)	OK: FDP_ACF.1/JCRMI
FDP_ACF.1/ADEL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/ADEL, FMT_MSA.3/ADEL
FDP_ACF.1/CMGR	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/CMGR, FMT_MSA.3/CMGR
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL
FDP_ACF.1/JCRMI	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/JCRMI, FMT_MSA.3/JCRMI
FDP_ACF.1.1/ EXT_MEM	(FDP_ACC.1) and (FMT_MSA.3)	OK: ***FDP_ACC.1.1/EXT_MEM, FMT_MSA.3.1/EXT_MEM***
FDP_IFC.1/JCRMI	(FDP_IFF.1)	OK: FDP_IFF.1/JCRMI
FDP_IFC.1/JCVM	(FDP_IFF.1)	OK: FDP_IFF.1/JCVM
FDP_IFC.2/BCV	(FDP_IFF.1)	OK: FDP_IFF.2/BCV
FDP_IFC.2/CM	(FDP_IFF.1)	OK: FDP_IFF.1/CM
FDP_IFF.1/CM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/CM, FMT_MSA.3/CM
FDP_IFF.1/JCRMI	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCRMI, FMT_MSA.3/JCRMI
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCVM, FMT_MSA.3/FIREWALL
FDP_IFF.2/BCV	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/BCV, FMT_MSA.3/BCV
FDP_ITC.2	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1) and (FPT_TDC.1)	OK: FPT_TDC.1, FDP_IFC.2/CM, FTP_ITC.1/CM
FDP_RIP.1	None	OK
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM
FDP_SDI.2	None	OK
FDP_UIT.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1)	OK: FDP_IFC.2/CM, FTP_ITC.1/CM
FIA_ATD.1/AID	None	OK
FIA_UID.1/CM	None	OK
FIA_UID.1/CMGR	None	OK
FIA_UID.2.1/AID	None	OK
FIA_USB.1	(FIA_ATD.1)	OK: FIA_ATD.1/AID

SFR	Dependency	Status
FMT_MSA.1/ADEL	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/ADEL, FMT_SMR.1/ADEL, ***FMT_SMF.1.1/ADEL***
FMT_MSA.1/BCV	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/BCV, FMT_SMR.1/BCV, ***FMT_SMF.1.1/BCV***
FMT_MSA.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/CM, FMT_SMR.1/CM, *FMT_SMF.1.1/CM*
FMT_MSA.1/CMGR	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.1/CMGR, FMT_SMR.1/CMGR, *FMT_SMF.1.1/CMGR*
FMT_MSA.1/EXPORT FMT_MSA.1/JCRMI FMT_MSA.1/REM-REFS	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.1/JCRMI, FMT_SMR.1/JCRMI, ***FMT_SMF.1.1/ADEL***, ***FMT_SMF.1.1/JCRMI***
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MSA.1.1/EXT_MEM,	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: ***FDP_ACC.1.1/EXT_MEM***, ***FMT_SMR.1.1/JCRE***, ***FMT_SMF.1.1/EXT_MEM***
FMT_MSA.2/JCRE	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.2/BCV	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_IFC.2/BCV, FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/ADEL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/ADEL, FMT_SMR.1/ADEL
FMT_MSA.3/BCV	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/CM	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CM, FMT_SMR.1/CM
FMT_MSA.3/CMGR	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CMGR, FMT_SMR.1/CMGR

SFR	Dependency	Status
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.3/JCRMI	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.3.1/EXT_MEM	(FMT_MSA.1) and (FMT_SMR.1)	OK: ***FMT_MSA.1.1/EXT_MEM***, ***FMT_SMR.1.1/JCRE***
FMT_MTD.1/JCRE	(FMT_SMR.1) and (FMT_SMF.1)	OK: FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***
FMT_MTD.3	(ADV_SPM.1) and (FMT_MTD.1)	OK: FMT_MTD.1/JCRE
FMT_REV.1/JCRMI	(FMT_SMR.1)	OK: FMT_SMR.1/JCRMI
FMT_SMR.1/ADEL	(FIA_UID.1)	KO: FIA_UID.1
FMT_SMR.1/BCV	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/CM	(FIA_UID.1)	OK: FIA_UID.1/CM
FMT_SMR.1/CMGR	(FIA_UID.1)	OK: FIA_UID.1/CMGR
FMT_SMR.1/JCRE	(FIA_UID.1)	OK:***FIA_UID.2.1/AID***
FMT_SMR.1/Installer	(FIA_UID.1)	KO: FIA_UID.1
FMT_SMR.1/JCRMI	(FIA_UID.1)	OK: ***FIA_UID.2.1/AID***
FPR_UNO.1	None	OK
FPT.PHP.3/SCP	None	OK
FPT_AMT.1/SCP	None	OK
FPT_FLS.1/ADEL	(ADV_SPM.1)	OK
FPT_FLS.1/Installer	(ADV_SPM.1)	OK
FPT_FLS.1/JCS	(ADV_SPM.1)	OK
FPT_FLS.1/ODEL	(ADV_SPM.1)	OK
FPT_FLS.1/SCP	(ADV_SPM.1)	OK
FPT_RCV.3/Installer	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK: FPT_TST.1
FPT_RCV.3/SCP	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK:FPT_TST.1
FPT_RCV.4/SCP	(ADV_SPM.1)	OK
FPT_RVM.1	None	OK
FPT_RVM.1/SCP	None	OK
FPT_SEP.1	None	OK
FPT_SEP.1/SCP	None	OK
FPT_TDC.1	None	OK
FPT_TST.1	(FPT_AMT.1)	OK: FPT_AMT.1/SCP
FRU_FLT.1/SCP	(FPT_FLS.1)	OK: FPT_FLS.1/SCP
FRU_RSA.1/Installer	None	OK
FRU_RSA.1/BCV	None	OK
FTP_ITC.1/CM	None	OK

Table 19: Functional Requirement Dependencies (Java Card System Standard 2.2)

FAU_SAA.1 Potential violation analysis is used to specify the set of auditable events whose occurrence or accumulated occurrence held to indicate a potential violation of the TSP, and any rules to be used to perform the violation

analysis. The dependency of FAU_ARP.1/JCS on this functional requirement assumes that a "potential security violation" is an audit event indicated by the FAU_SAA.1 component. The events listed in FAU_ARP.1/JCS are, on the contrary, merely self-contained ones (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The *Java Card VM* or other components of the TOE detect these events during their usual working order. Thus, in principle there would be no applicable audit recording in this framework. Moreover, no specification of one such recording is provided elsewhere. Therefore no set of auditable events could possibly be defined.

FIA_UID.1

This is required by the component ***FMT_SMR.1*** of the group *InstG*. However, the role installer defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The installer does not "identify" itself with respect to the TOE, but is a part of it. Thus, here it is claimed that this dependency can be left out. The reader may notice that the role is required because of the SFRs on management of TSF data and security attributes, essentially those of the firewall policy.

This is also required by the component ***FMT_SMR.1*** in group *ADELG*. See the explanation in the paragraph above (the role in this case is applet deletion manager).

This is also required by the component ***FMT_SMR.1*** of the group *BCVG*. However, the role bytecode verifier defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The bytecode verifier does not "identify" itself with respect to the TOE, furthermore, it is part of the IT environment. Thus, here it is claimed that this dependency can be left out.

6.2.3.4 Rationale for Strength of Function Medium

The minimum strength of function level required is **SOF-medium**.

The TOE is intended to operate in open environments, where attackers can easily exploit vulnerabilities. According to the claimed intended usage of the TOE, it is very likely that it may represent a significant value and then constitute an attractive target for attacks. In some malicious usages of the TOE the statistical or probabilistic mechanisms in the TOE, for instance, may be subjected to analysis and attack in the normal course of operation. A medium strength of function seems to be the reasonable minimum level for cards hosting sensitive applications. It shall probably be the case, as it is frequent nowadays, that the required strength of function level will be high in, for instance, banking or electronic signature applications. Considering that Java Card technology-based products may also address other less security sensitive contexts, and furthermore, that the resistance of the mechanisms mentioned above to attacks with high potential is hard to be achieved and demonstrated, the choice of a high strength of function requirement is left to the card issuer depending on the intended usage of the product. Thus, in this protection profile it has been chosen a protection against moderate attack potential as the minimal level for those multi-applicative cards.

The strength of function level medium is consistent with the vulnerability analysis level that has been specified (***AVA_VLA.3).***

6.2.3.5 Rationale for Assurance Level EAL4 augmented

The assurance level for this protection profile is EAL4 augmented. Augmentation results from the selection of the components ***AVA_VLA.3*** and ***ADV_IMP.2.***

6.2.3.5.1 Rationale for Assurance Level EAL4

EAL4 allows a developer to attain a reasonably high assurance level without the need for highly specialized processes and practices. It corresponds to a white box analysis and it can be considered as a reasonable level that can be applied to an existing product line without undue expense and complexity.

6.2.3.5.2 Rationale for Augmentation

The evaluation of the TOE may be performed, for instance, because the product hosts one or several sensitive applications, such as financial and health recording ones, which contain, represent, or provide access to valuable assets. In addition to that the TOE may not be directly under the control of trained and dedicated administrators.

AVA_VLA.3

As a result, it is imperative that the TOE vulnerabilities to be reviewed be drawn from a systematic search rather than strictly a manufacturer prepared identification list. Component ***AVA_VLA.3*** requires that such a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***. There might be scenarios, for example if the TOE is intended to stay in a hostile environment for long periods of time, or if the applications are considered to be highly sensitive, that would justify a further augmentation by requiring the component ***AVA_VLA.4.*** This latter component dictates that the TOE must be shown to be resistant to penetration attacks performed by attackers possessing a high attack potential. The choice of augmenting the assurance level using the component ***AVA_VLA.4*** is left to the card issuer.

AVA_VLA.3 has the following dependencies:

- ***ADV_FSP.1*** Informal functional specification
- ***ADV_HLD.2*** Security enforcing high-level design
- ***ADV_IMP.1*** Subset of the implementation of the TSF
- ***ADV_LLD.1*** Descriptive low-level design
- ***AGD_ADM.1*** Administrator guidance
- ***AGD_USR.1*** User guidance

All of these are met or exceeded in the EAL4 assurance package.

ADV_IMP.2

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement.

The assurance component ***ADV_IMP.2*** has been chosen because the evaluation of the TOE must ensure that its security functional requirements are completely and accurately addressed by the implementation representation of the TSF.

ADV_IMP.2 has the following dependencies:

ADV_LLD.1 Descriptive low-level design

ADV_RCR.1 Informal correspondence demonstration

ALC_TAT.1 Well-defined development tools

All of these are met or exceeded in the EAL4 assurance package.

6.2.3.6 Internal Consistency and Mutual Support

The purpose of this part of the Protection Profile rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given to this because:

The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies are satisfied).

The dependencies analysis for the functional requirements described in the section "Security Functional Requirements Dependencies" demonstrates mutual support and internal consistency between the functional requirements. That analysis also shows that the dependencies between functional and assurance requirements are also satisfied.

6.2.4 Defensive Configuration

6.2.4.1 TOE Security Requirements Rationale

In the context of this rationale the *FIREWALL access control policy* is the one specified in the group *LCG* (§5.1.6). The references to the components *FDP_ACC.2/FIREWALL*, *FDP_ACF.1/FIREWALL* and *FMT_MSA.1/JCRE* must be understood as denoting the definitions of those components as provided in the group *LCG*.

This rationale for this configuration is almost the same than the one defined for the **Java Card System Standard 2.2** configuration. There are two main differences:

1. The configuration **Defensive** has no security objective *O.LOAD*. The packages loaded post-issuance are verified on card. Therefore there shall be no reference to the SFRs of the group *CarG* (§5.1.8).

2. The configuration **Defensive** is the only one to have as security objective *O.1205HVERIFICATION*. Therefore there shall be references to the SFRs of the group *BCVG* (§5.1.3).

Note: the differences between the **Defensive** and the **Java Card System Standard 2.2** configurations have been underlined in the following rationale.

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

APPLET MANAGEMENT	
O.INSTALL	This objective specifies that installation of *applets* must be secure. Security attributes of installed data are under the control of the *FIREWALL* access control *policy* (*FDP_ITC.2*), and the TSFs are protected against possible failures of the *installer* (*FPT_FLS.1/Installer*, *FPT_RCV.3*).
O.1186HDELETION	This objective specifies that *applet* and *package* deletion must be secure. The non-introduction of security holes is ensured by the *ADEL access control* ***policy*** (*FDP_ACC.2/ADEL, FDP_ACF.1/ADEL)*. The integrity and confidentiality of data that does not belong to the deleted *applet* or *package* is a by-product of this policy as well. Non-accessibility of deleted data is met by *FDP_RIP.1/ADEL* and the TSFs are protected against possible failures of the deletion procedures (*FPT_FLS.1/ADEL*, *FPT_RCV.3*

(see application note)). The functional requirements of the class ***FMT*** included in the group *ADELG* also contribute to meet this objective.

EXECUTION

O.OPERATE

The TOE is protected in various ways against *applets*' actions (***FPT_RVM.1***, ***FPT_SEP.1***, *FPT_TDC.1*), the ***FIREWALL*** *access control policy* (*FDP_ACC.2*, *FDP_ACF.1*), and is able to detect and block various failures or security violations during usual working (*FPT_FLS.1*, *FAU_ARP.1*). Startup of the TOE is covered by *FPT_TST.1*, and indirectly by *FPT_AMT.1* (the latter is defined in group *SCPG* §5.1.9).

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

Almost every objective and/or functional requirement indirectly contributes to this one too, in particular the EXTERNAL MEMORY access control policy (*FDP_ACC.1*, *FDP_ACF.1*), if the TOE implements external memory features.

O.RESOURCES

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

O.FIREWALL

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

O.NATIVE

The *Java Card VM* is the machine running the bytecode of the *applets* (***FPT_RVM.1***). These can only be linked with API methods or other *packages* already on the card. This objective mainly relies on the environmental objectives ***OE.NATIVE*** and in the requirement of secure security attributes expressed by the component ***FMT_MSA.2*** of the group *BCVG* (§5.1.3) (see application note at pp. 90).

O.REALLOCATION

The security objective is satisfied by *FDP_RIP.1*, which imposes that the contents of the re-allocated block shall always be cleared before delivering the block. If the block is used to store the local variables of a newly allocated frame, then the ***TYPING information flow control policy*** of the group *BCVG* (*FDP_IFC.2/BCV*, *FDP_IFF.2/BCV*) also contributes to satisfy this objective by ensuring that the local variable is never read before being assigned with an initial value.

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

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

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

SERVICES

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

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

Transactions are provided to *applet*s as Java Card class libraries.

O.CIPHER This objective is directly related to ***FCS_CKM.1***, ***FCS_CKM.2***, ***FCS_CKM.3***, ***FCS_CKM.4*** and ***FCS_COP.1***. Another important SFR is ***FPR_UNO.1***, the observation of the cryptographic operations may be used to disclose the keys.

The associated security functions are not described herein. They are provided to *applet*s as Java Card class libraries (see the class **`javacardx.crypto.Cipher`** and the package **`javacard.security`**).

O.PIN-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements.API classes implement the security functions behind these. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the objects.

O.BIO-MNGT This objective is ensured by *FDP_RIP.1*, ***FPR_UNO.1***, *FDP_ROL.1* and ***FDP_SDI.2*** functional requirements. The applets that manage biometric templates rely on the security functions that implement these requirements. The *firewall* security functions (*FDP_ACC.2, FDP_ACF.1)* shall protect the access to private and internal data of the templates. Note that the objective applies only to configurations including the javacardx.biometry package defined in [JCAPI222].

O.KEY-MNGT This relies on the same functional requirements as *O.CIPHER*, plus *FDP_RIP.1* and ***FDP_SDI.2*** as well.

O.REMOTE The access to the TOE's internal data and the flow of information from the card to the *CAD* required by the *Java Card RMI* service is under control of the *Java Card RMI access control policy* (*FDP_ACC.2/JCRMI*, *FDP_ACF.1/JCRMI*) and the ***JCRMI*** *information flow* control policy (*FDP_IFC.1/JCRMI*, *FDP_IFF.1/JCRMI*). The functional requirements of the class ***FMT*** included in the group *RMIG* also contribute to meet this objective.*O.EXT_MEM*_____The Java Card System memory is protected against applet's tentatives of unauthorized access through the external memory facilities by the *EXTERNAL MEMORY access control policy* (***FDP_ACC.1.1/EXT_MEM***, ***FDP_ACF.1.1/ EXT_MEM***), which first controls the accessible address space, then controls the effective read and write operations.

OBJECT DELETION

O.1189HOBJ-DELETION This objective specifies that deletion of objects is secure. The objective is met by the functional requirements *FDP_RIP.1/ODEL* and *FPT_FLS.1/ODEL*.

INTEGRITY, CONFIDENTIALITY AND CORRECT EXECUTION

O.1205HVERIFICATION This objective is directly met by the ***TYPING information flow control policy*** (*FDP_IFC.2/BCV, FDP_IFF.2/BCV*) and the functional requirements of the group *BCVG* (§5.1.3).

	FAU_ARP.1	FCS_CKM.1	FCS_CKM.2	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.1	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FDP_ROL.1	FDP_SDI.2	FIA_ATD.1	FIA_UID.2	FIA_USB.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_MTD.3	FMT_SMR.1	FMT_SMF.1	FPR_UNO.1	FPT_FLS.1	FPT_RVM.1	FPT_SEP.1	FPT_TDC.1	FPT_TST.1
O.ALARM	X																									X				
O.CIPHER		X	X	X	X	X																			X					
O.FIREWALL								X	X	X	X							X	X	X	X	X	X	X			X	X		
O.KEY-MNGT		X	X	X	X	X						X		X											X					
O.NATIVE																			X								X			
O.OPERATE	X							X	X				X		X		X									X	X	X	X	X
O.PIN-MNGT								X	X			X	X	X											X					
O.BIO-MNGT								X	X			X	X	X																
O.RESOURCES	X												X								X	X	X			X	X			
O.SID															X	X	X	X		X	X	X		X			X	X		
O.TRANSACTION												X	X																	
O.SHRD_VAR_CONFID										X	X	X																		
O.SHRD_VAR_INTEG										X	X																			
O.REALLOCATION												X																		
O.EXT_MEM							X		X									X	X	X			X	X						

	FDP_ITC.2	FPT_FLS.1	FPT_RCV.3	FRU_RSA.1
O.INSTALL	X	X	X	
O.SID	X			
O.OPERATE	X		X	
O.RESOURCES			X	X
O.FIREWALL	X			

	FDP_ACC.2	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_RIP.1	FMT_MSA.1	FMT_MSA.3	FMT_REV.1	FMT_SMR.1	FPT_FLS.1	FPT_RCV.3
O.DELETION	X	X			X	X	X		X	X	X
O.OBJ-DELETION					X					X	
O.REMOTE	X	X	X	X		X	X	X	X		
O.FIREWALL								X			

	FDP_IFC.2	FDP_IFF.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_SMR.1	FRU_RSA.1
O.REALLOCATION	X	X	X		X		
O.VERIFICATION	X	X	X	X	X	X	X

Table 20: Security requirements rationale for the Defensive Configuration

6.2.4.2 IT Environment Security Requirements Rationale

The environmental objective *OE.CARD-MANAGEMENT*, which is satisfied by IT procedural means, is met by the SFRs of the group *CMGRG* (§5.1.10).

All the security functional requirements to which this section makes reference from now on are those specified in the group *SCPG* (§5.1.9).

The components *FPT_RCV.3* and *FPT_RCV.4* are used to support the objective *OE.1212H*SCP.*SUPPORT* and *OE.1210H*SCP.*RECOVERY* to assist the TOE to recover in the event of a power failure or signal loss. If the RF signal is lost, the power fails or the card is withdrawn prematurely from the CAD the operation of the TOE may be interrupted leaving the TOE in an inconsistent state.

OE.SCP.RECOVERY This objective is met by the components *FPT_FLS.1*, *FPT_RCV.3* and *FRU_FLT.1*.

OE.SCP.SUPPORT This objective is met by the components *FPT_SEP.1* (no bypassing TSF), *FPT_AMT.1*, *FPT_RCV.3*, *FPT_RCV.4* and *FPT_RVM.1*.

OE.SCP.IC This objective is met by the component *FPT_PHP.3*.

	FPT_AMT.1	FPT_FLS.1	FPT_PHP.3	FPT_RCV.3	FPT_RCV.4	FPT_RVM.1	FPT_SEP.1	FRU_FLT.1
OE.SCP.RECOVERY		X		X				X
OE.SCP.SUPPORT	X			X	X	X	X	
OE.SCP.IC			X					

Table 21: Security requirements rationale for the group SCPG

6.2.4.3 Security Functional Requirements Dependencies

The TOE assurance requirements dependencies for level EAL4 are completely fulfilled.

The functional requirements dependencies for the TOE are not completely fulfilled. The KOs in the following table corresponds to unsatisfied dependencies that are explained and justified in the rationale that appears below the table.

SFR	Dependency	Status
FAU_ARP.1/JCS	(FAU_SAA.1)	KO: FAU_SAA.1 is not satisfied
FCS_CKM.1	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.2, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.2	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.3	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FCS_CKM.4	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FMT_MSA.2)	OK: FCS_CKM.1, FMT_MSA.2/JCRE
FCS_COP.1	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1) and (FCS_CKM.4) and (FMT_MSA.2)	OK: FCS_CKM.1, FCS_CKM.4, FMT_MSA.2/JCRE
FDP_ACC.1/CMGR	(FDP_ACF.1)	OK: FDP_ACF.1/CMGR
FDP_ACC.2/ADEL	(FDP_ACF.1)	OK:FDP_ACF.1/ADEL
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	OK: FDP_ACF.1/FIREWALL
FDP_ACC.2/JCRMI	(FDP_ACF.1)	OK: FDP_ACF.1/JCRMI
FDP_ACC.1.1/EXT_MEM	(FDP_ACF.1)	OK: ***FDP_ACF.1.1/ EXT_MEM***
FDP_ACF.1/ADEL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/ADEL, FMT_MSA.3/ADEL
FDP_ACF.1/CMGR	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.1/CMGR, FMT_MSA.3/CMGR

SFR	Dependency	Status
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	OK: FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL
FDP_ACF.1/JCRMI	(FDP_ACC.1) and (FMT_MSA.3)	OK FDP_ACC.2/JCRMI, FMT_MSA.3/JCRMI
FDP_ACF.1.1/EXT_MEM	(FDP_ACC.1) and (FMT_MSA.3)	OK: ***FDP_ACC.1.1/EXT_MEM***, ***FMT_MSA.3.1/EXT_MEM***
FDP_IFC.1/JCRMI	(FDP_IFF.1)	OK: FDP_IFF.1/JCRMI
FDP_IFC.1/JCVM	(FDP_IFF.1)	OK: FDP_IFF.1/JCVM
FDP_IFC.2/BCV	(FDP_IFF.1)	OK: FDP_IFF.2/BCV
FDP_IFF.1/JCRMI	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCRMI, FMT_MSA.3/JCRMI
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.1/JCVM, FMT_MSA.3/FIREWALL
FDP_IFF.2/BCV	(FDP_IFC.1) and (FMT_MSA.3)	OK: FDP_IFC.2/BCV, FMT_MSA.3/BCV
FDP_ITC.2	(FDP_ACC.1 or FDP_IFC.1) and (FTP_ITC.1 or FTP_TRP.1) and (FPT_TDC.1)	OK: FPT_TDC.1, FDP_IFC.2/BCV, KO: FTP_ITC.1 or FTP_TRP.1
FDP_RIP.1	None	OK
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM
FDP_SDI.2	None	OK
FIA_ATD.1/AID	None	OK
FIA_UID.1/CMGR	None	OK
FIA_UID.2.1/AID	None	OK
FIA_USB.1	(FIA_ATD.1)	OK: FIA_ATD.1/AID
FMT_MSA.1/ADEL	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/ADEL, FMT_SMR.1/ADEL, ***FMT_SMF.1.1/ADEL***
FMT_MSA.1/BCV	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.2/BCV, FMT_SMR.1/BCV, ***FMT_SMF.1.1/BCV***
FMT_MSA.1/CMGR	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.1/CMGR, FMT_SMR.1/CMGR, *FMT_SMF.1.1/CMGR*
FMT_MSA.1/EXPORT FMT_MSA.1/JCRMI FMT_MSA.1/REM-REFS	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_IFC.1/JCRMI, FMT_SMR.1/JCRMI, ***FMT_SMF.1.1/ADEL***, ***FMT_SMF.1.1/JCRMI***
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE***, ***FMT_SMF.1.1/ADEL***

SFR	Dependency	Status
FMT_MSA.1.1/EXT_MEM,	(FDP_ACC.2 or FDP_IFC.1) and (FMT_SMR.1) and (FMT_SMF.1)	OK: ***FDP_ACC.1.1/EXT_MEM, FMT_SMR.1.1/JCRE, FMT_SMF.1.1/EXT_MEM***
FMT_MSA.2/JCRE	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.2/BCV	(ADV_SPM.1) and (FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	OK: FDP_IFC.2/BCV, FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/ADEL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/ADEL, FMT_SMR.1/ADEL
FMT_MSA.3/BCV	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/BCV, FMT_SMR.1/BCV
FMT_MSA.3/CMGR	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/CMGR, FMT_SMR.1/CMGR
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRE, FMT_SMR.1/JCRE
FMT_MSA.3/JCRMI	(FMT_MSA.1) and (FMT_SMR.1)	OK: FMT_MSA.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.3.1/EXT_MEM	(FMT_MSA.1) and (FMT_SMR.1)	OK: ***FMT_MSA.1.1/EXT_MEM, FMT_SMR.1.1/JCRE***
FMT_MTD.1/JCRE	(FMT_SMR.1) and (FMT_SMF.1)	OK: FMT_SMR.1/JCRE, ***FMT_SMF.1.1/JCRE, FMT_SMF.1.1/ADEL***
FMT_MTD.3	(ADV_SPM.1) and (FMT_MTD.1)	OK: FMT_MTD.1/JCRE
FMT_REV.1/JCRMI	(FMT_SMR.1)	OK: FMT_SMR.1/JCRMI
FMT_SMR.1/ADEL	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/BCV	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/CMGR	(FIA_UID.1)	OK: FIA_UID.1/CMGR
FMT_SMR.1/JCRE	(FIA_UID.1)	OK: ***FIA_UID.2.1/AID***
FMT_SMR.1/Installer	(FIA_UID.1)	KO: (FIA_UID.1)
FMT_SMR.1/JCRMI	(FIA_UID.1)	OK: ***FIA_UID.2.1/AID***
FPR_UNO.1	None	OK
FPT.PHP.3/SCP	None	OK
FPT_AMT.1/SCP	None	OK
FPT_FLS.1/ADEL	(ADV_SPM.1)	OK
FPT_FLS.1/Installer	(ADV_SPM.1)	OK
FPT_FLS.1/JCS	(ADV_SPM.1)	OK
FPT_FLS.1/ODEL	(ADV_SPM.1)	OK
FPT_FLS.1/SCP	(ADV_SPM.1)	OK

SFR	Dependency	Status
FPT_RCV.3/Installer	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK: FPT_TST.1
FPT_RCV.3/SCP	(FPT_TST.1) and (AGD_ADM.1) and (ADV_SPM.1)	OK:FPT_TST.1
FPT_RCV.4/SCP	(ADV_SPM.1)	OK
FPT_RVM.1	None	OK
FPT_RVM.1/SCP	None	OK
FPT_SEP.1	None	OK
FPT_SEP.1/SCP	None	OK
FPT_TDC.1	None	OK
FPT_TST.1	(FPT_AMT.1)	OK: FPT_AMT.1/SCP
FRU_FLT.1/SCP	(FPT_FLS.1)	OK: FPT_FLS.1/SCP
FRU_RSA.1/BCV	None	OK
FRU_RSA.1/Installer	None	OK

Table 22: Functional Requirement Dependencies (Defensive)

FAU_SAA.1

Potential violation analysis is used to specify the set of auditable events whose occurrence or accumulated occurrence held to indicate a potential violation of the TSP, and any rules to be used to perform the violation analysis. The dependency of FAU_ARP.1/JCS on this functional requirement assumes that a "potential security violation" is an audit event indicated by the FAU_SAA.1 component. The events listed in FAU_ARP.1/JCS are, on the contrary, merely self-contained ones (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The *Java Card VM* or other components of the TOE detect these events during their usual working order. Thus, in principle there would be no applicable audit recording in this framework. Moreover, no specification of one such recording is provided elsewhere. Therefore no set of auditable events could possibly be defined.

FTP_ITC.1 or ***FTP_TRP.1***

Import from outside TSF control defines the mechanisms for introduction of user data into the TOE such that it has appropriate security attributes and is appropriately protected. The dependency of ***FDP_ITC.2*** on one of these components is not justified in the presence of on-card bytecode verification.

FIA_UID.1

This is required by the component ***FMT_SMR.1*** of the group *InstG*. However, the role installer defined in this component is attached to an IT security function rather than to a "user" of the CC terminology. The installer does not "identify" itself with respect to the TOE, but is a part of it. Thus, here it is claimed that this dependency can be left out. The reader may notice that the role is required because of the SFRs on management of TSF data and security attributes, essentially those of the firewall policy.

This is also required by the component ***FMT_SMR.1*** in groups *ADELG* and *BCVG*. See the explanation in the paragraph above (the roles in this case are applet deletion manager and bytecode verifier).

6.2.4.4 Rationale for Strength of Function Medium

The minimum strength of function level required is **SOF-medium**.

The TOE is intended to operate in open environments, where attackers can easily exploit vulnerabilities. According to the claimed intended usage of the TOE, it is very likely that it may represent a significant value and then constitute an attractive target for attacks. In some malicious usages of the TOE the statistical or probabilistic mechanisms in the TOE, for instance, may be subjected to analysis and attack in the normal course of operation. A medium strength of function seems to be the reasonable minimum level for cards hosting sensitive applications. It shall probably be the case, as it is frequent nowadays, that the required strength of function level will be high in, for instance, banking or electronic signature applications. Considering that Java Card technology-based products may also address other less security sensitive contexts, and furthermore, that the resistance of the mechanisms mentioned above to attacks with high potential is hard to be achieved and demonstrated, the choice of a high strength of function requirement is left to the card issuer depending on the intended usage of the product. Thus, in this protection profile, a protection against moderate attack potential has been chosen as the minimal level for those multi-applicative cards.

The strength of function level medium is consistent with the vulnerability analysis level that has been specified (***AVA_VLA.3).***

6.2.4.5 Rationale for Assurance Level EAL4 augmented

The assurance level for this protection profile is EAL4 augmented. Augmentation results from the selection of the components ***AVA_VLA.3*** and ***ADV_IMP.2.***

6.2.4.5.1 Rationale for Assurance Level EAL4

EAL4 allows a developer to attain a reasonably high assurance level without the need for highly specialized processes and practices. It corresponds to a white box analysis and it can be considered as a reasonable level that can be applied to an existing product line without undue expense and complexity.

6.2.4.5.2 Rationale for Augmentation

The evaluation of the TOE may be performed, for instance, because the product hosts one or several sensitive applications, such as financial and health recording ones, which contain, represent, or provide access to valuable assets. In addition to that the TOE may not be directly under the control of trained and dedicated administrators.

AVA_VLA.3

As a result, it is imperative that the TOE vulnerabilities to be reviewed be drawn from a systematic search rather than strictly a manufacturer prepared identification list. Component ***AVA_VLA.3*** requires that such a systematic search for vulnerabilities be documented and presented. This provides a significant increase in the consideration of vulnerabilities over that provided by ***AVA_VLA.2***. There might be scenarios, for example if the TOE is intended to stay in a hostile environment for long periods of time, or if the applications are considered to be highly sensitive, that would justify a further augmentation by requiring the component ***AVA_VLA.4.*** This latter component dictates that the TOE must be shown to be resistant to penetration attacks performed by attackers possessing a high attack potential. The choice of augmenting the assurance level using the component ***AVA_VLA.4*** is left to the card issuer.

AVA_VLA.3 has the following dependencies:

ADV_FSP.1 Informal functional specification

ADV_HLD.2 Security enforcing high-level design

ADV_IMP.1 Subset of the implementation of the TSF

ADV_LLD.1 Descriptive low-level design

AGD_ADM.1 Administrator guidance

AGD_USR.1 User guidance

All of these are met or exceeded in the EAL4 assurance package.

ADV_IMP.2

The implementation representation is used to express the notion of the least abstract representation of the TSF, specifically the one that is used to create the TSF itself without further design refinement.

The assurance component ***ADV_IMP.2*** has been chosen because the evaluation of the TOE must ensure that its security functional requirements are completely and accurately addressed by the implementation representation of the TSF.

ADV_IMP.2 has the following dependencies:

ADV_LLD.1 Descriptive low-level design

ADV_RCR.1 Informal correspondence demonstration

ALC_TAT.1 Well-defined development tools

All of these are met or exceeded in the EAL4 assurance package.

6.2.4.6 Internal Consistency and Mutual Support

The purpose of this part of the Protection Profile rationale is to show that the security requirements are mutually supportive and internally consistent. No detailed analysis is given to this because:

The dependencies analysis for the additional assurance components in the previous section has shown that the assurance requirements are mutually supportive and internally consistent (all the dependencies are satisfied).

The dependencies analysis for the functional requirements described in the section "Security Functional Requirements Dependencies" demonstrates mutual support and internal consistency between the functional requirements. That analysis also shows that the dependencies between functional and assurance requirements are also satisfied.

7 APPENDIX: A UNIFIED VIEW OF CONFIGURATIONS

This section provides an all-embracing presentation of the security environment, security objectives and functional requirements of the configurations defined in this document. The tables included below not only make explicit the contents proper of each configuration but also reflects the differences between the configurations.

Assets are common to all configurations. Those corresponding to **User data** are**: D.APP_CODE, D.APP_C_DATA, D.APP_I_DATA, D.PIN** and **D.APP_KEYs**. **D.BIO** is also user data in version 2.2.2 of the Java Card platform. Those corresponding to **TSF data** are: **D.JCS_CODE, D.JCS_DATA, D.SEC_DATA, D.API_DATA, D.CRYPTO** and **D.JCS_KEYs**.

The configurations' assumptions are displayed in Table 23.

Assumption	Minimal	Standard 2.1.1	Standard 2.2	Defensive
A.NATIVE	X	X	X	X
A.NO-INSTALL	X			
A.NO-DELETION	X			
A.DELETION		X		
A.APPLET		X	X	
A.VERIFICATION	X	X	X	

Table 23: Assumptions of Configurations

The threats to the assets against which specific protection is required within the configurations or their environments are displayed in Table 24. The post-issuance installation of applets introduces one threat (*T.INSTALL*), and two more (*T.INTEG-APPLI-CODE.2, T.INTEG-APPLI-DATA.2)* in the case that bytecode verification is performed off-card. Thereby the absence of the latter two threats in the Java Card System 2.2 Defensive configuration.

Threat	Minimal	Standard 2.1.1	Standard 2.2	Defensive
T.PHYSICAL	X	X	X	X
T.CONFID-JCS-CODE	X	X	X	X
T.CONFID-APPLI-DATA	X	X	X	X
T.CONFID-JCS-DATA	X	X	X	X
T.INTEG-APPLI-CODE	X	X	X	X
T.INTEG-JCS-CODE	X	X	X	X
T.INTEG-APPLI-DATA	X	X	X	X
T.INTEG-JCS-DATA	X	X	X	X
T.SID.1	X	X	X	X
T.SID.2	X	X	X	X
T.EXE-CODE.1	X	X	X	X
T.EXE-CODE.2	X	X	X	X
T.NATIVE	X	X	X	X
T.RESOURCES	X	X	X	X
T.INTEG-APPLI-CODE.2		X	X	
T.INTEG-APPLI-DATA.2		X	X	
T.INSTALL		X	X	X
T.EXE-CODE-REMOTE			X	X
T.DELETION			X	X
T.OBJ-DELETION			X	X

Table 24: Threats of Configurations

There is only one organizational security policy defined in this document, *OSP.1131HVERIFICATION*, which applies for both the **Java Card System Standard 2.1.1** and the **Java Card System Standard 2.2** configurations.

Each configuration determines a particular TOE. Table 25 lists the security objectives addressed by each of those TOEs. The configuration that includes an on -card bytecode verifier is the only one to have the verification of the bytecodes of a package as a security objective. The addressing of that objective is the difference between the Defensive and the Standard 2.2 configurations.

TOE security objective	Minimal	Standard 2.1.1	Standard 2.2	Defensive
O.SID	X	X	X	X
O.OPERATE	X	X	X	X
O.RESOURCES	X	X	X	X
O.FIREWALL	X	X	X	X
O.NATIVE	X	X	X	X
O.REALLOCATION	X	X	X	X
O.SHRD_VAR_CONFID	X	X	X	X
O.SHRD_VAR_INTEG	X	X	X	X
O.ALARM	X	X	X	X
O.TRANSACTION	X	X	X	X
O.CIPHER	X	X	X	X
O.PIN-MNGT	X	X	X	X
O.KEY-MNGT	X	X	X	X
*O.1177H*INSTALL		X	X	X
O.LOAD		X	X	
O.1186HDELETION			X	X
O.OBJ-DELETION			X	X
O.REMOTE			X	X
*O.BIO-*MNGT			X	X
O.EXT_MEM			X	X
O.VERIFICATION				X

Table 25: TOE Security Objectives of Configurations

Table 26 displays the security objectives to be achieved by the environment associated to each TOE configuration.

Environment security objective	Minimal	Standard 2.1.1	Standard 2.2	Defensive
OE.NATIVE	X	X	X	X
OE.SCP.RECOVERY	X	X	X	X
OE.SCP.SUPPORT	X	X	X	X
OE.SCP.IC	X	X	X	X
OE.NO-DELETION	X			
OE.NO-INSTALL	X			
OE.VERIFICATION	X	X	X	
OE.APPLET		X	X	
OE.CARD-MANAGEMENT	X	X	X	X

Table 26: Security objectives for the environment of Configurations

Finally, Table 27 makes explicit the relation between SFRs, and the groups to which they belong, and the several configurations defined in this document.

SFR	Group	Minimal	Standard 2.1.1	Standard 2.2	Defensive
FAU_ARP.1/JCS	*CoreG*	X	X	X	X
FCS_CKM.1	*CoreG*	X	X	X	X
FCS_CKM.2	*CoreG*	X	X	X	X
FCS_CKM.3	*CoreG*	X	X	X	X
FCS_CKM.4	*CoreG*	X	X	X	X
FCS_COP.1	*CoreG*	X	X	X	X

SFR	Group	Minimal	Standard 2.1.1	Standard 2.2	Defensive
FDP_ACC.2/FIREWALL	*CoreG*	X	X		
FDP_ACF.1/FIREWALL	*CoreG*	X	X		
FDP_IFC.1/JCVM	*CoreG*	X	X	X	X
FDP_IFF.1/JCVM	*CoreG*	X	X	X	X
FDP_RIP.1/ABORT	*CoreG*	X	X	X	X
FDP_RIP.1/APDU	*CoreG*	X	X	X	X
FDP_RIP.1/bArray	*CoreG*	X	X	X	X
FDP_RIP.1/KEYS	*CoreG*	X	X	X	X
FDP_RIP.1/OBJECTS	*CoreG*	X	X	X	X
FDP_RIP.1/TRANSIENT	*CoreG*	X	X		
FDP_ROL.1/FIREWALL	*CoreG*	X	X	X	X
FDP_SDI.2	*CoreG*	X	X	X	X
FIA_ATD.1/AID	*CoreG*	X	X	X	X
FIA_UID.2/AID	*CoreG*	X	X	X	X
FIA_USB.1	*CoreG*	X	X	X	X
FMT_MSA.1/JCRE	*CoreG*	X	X		
FMT_MSA.2/JCRE	*CoreG*	X	X	X	X
FMT_MSA.3/FIREWALL	*CoreG*	X	X	X	X
FMT_MTD.1/JCRE	*CoreG*	X	X	X	X
FMT_MTD.3	*CoreG*	X	X	X	X
FMT_SMR.1/JCRE	*CoreG*	X	X	X	X
FPR_UNO.1	*CoreG*	X	X	X	X
FPT_FLS.1/JCS	*CoreG*	X	X	X	X
FPT_RVM.1	*CoreG*	X	X	X	X
FPT_SEP.1	*CoreG*	X	X	X	X
FPT_TDC.1	*CoreG*	X	X	X	X
FPT_TST.1	*CoreG*	X	X	X	X
FDP_ITC.2	*InstG*		X	X	X
FMT_SMR.1/Installer	*InstG*		X	X	X
FPT_FLS.1/Installer	*InstG*		X	X	X
FPT_RCV.3/Installer	*InstG*		X	X	X
FRU_RSA.1/Installer	*InstG*		X	X	X
FDP_IFC.2/BCV	*BCVG*	X	X	X	X
FDP_IFF.2/BCV	*BCVG*	X	X	X	X
FMT_MSA.1/BCV	*BCVG*	X	X	X	X
FMT_MSA.2/BCV	*BCVG*	X	X	X	X
FMT_MSA.3/BCV	*BCVG*	X	X	X	X
FMT_SMR.1/BCV	*BCVG*	X	X	X	X
FRU_RSA.1/BCV	*BCVG*	X	X	X	X
FDP_ACC.2/ADEL	*ADELG*			X	X
FDP_ACF.1/ADEL	*ADELG*			X	X
FMT_MSA.1/ADEL	*ADELG*			X	X
FMT_MSA.3/ADEL	*ADELG*			X	X
FMT_SMR.1/ADEL	*ADELG*			X	X
FDP_RIP.1/ADEL	*ADELG*			X	X
FPT_FLS.1/ADEL	*ADELG*			X	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/JCRMI	*RMIG*			X	X

SFR	Group	Minimal	Standard 2.1.1	Standard 2.2	Defensive
FMT_MSA.3/JCRMI	*RMIG*			X	X
FMT_REV.1/JCRMI	*RMIG*			X	X
FMT_SMR.1/JCRMI	*RMIG*			X	X
FDP_ACC.2/FIREWALL	*LCG*			X	X
FDP_ACF.1/FIREWALL	*LCG*			X	X
FMT_MSA.1/JCRE	*LCG*			X	X
FDP_RIP.1/TRANSIENT	*LCG*			X	X
FDP_RIP.1/ODEL	*ODELG*			X	X
FPT_FLS.1/ODEL	*ODELG*			X	X
FCO_NRO.2/CM	*CarG*		X	X	
FDP_IFC.2/CM	*CarG*		X	X	
FDP_IFF.1/CM	*CarG*		X	X	
FDP_UIT.1/CM	*CarG*		X	X	
FMT_MSA.1/CM	*CarG*		X	X	
FMT_MSA.3/CM	*CarG*		X	X	
FMT_SMR.1/CM	*CarG*		X	X	
FIA_UID.1/CM	*CarG*		X	X	
FTP_ITC.1/CM	*CarG*		X	X	
FPT_PHP.3/SCP	*SCPG*	X	X	X	X
FPT_AMT.1/SCP	*SCPG*	X	X	X	X
FPT_FLS.1/SCP	*SCPG*	X	X	X	X
FPT_RCV.3/SCP	*SCPG*	X	X	X	X
FPT_RCV.4/SCP	*SCPG*	X	X	X	X
FPT_RVM.1/SCP	*SCPG*	X	X	X	X
FPT_SEP.1/SCP	*SCPG*	X	X	X	X
FRU_FLT.1/SCP	*SCPG*	X	X	X	X
FDP_ACC.1/CMGR	*CMGRG*	X	X	X	X
FDP_ACF.1/CMGR	*CMGRG*	X	X	X	X
FIA_UID.1/CMGR	*CMGRG*	X	X	X	X
FMT_MSA.1/CMGR	*CMGRG*	X	X	X	X
FMT_MSA.3/CMGR	*CMGRG*	X	X	X	X
FMT_SMR.1/CMGR	*CMGRG*	X	X	X	X
FDP_ACC.1.1/EXT_MEM	*EMG*			X	X
FDP_ACF.1.1/ EXT_MEM	*EMG*			X	X
FMT_MSA.1.1/EXT_MEM	*EMG*			X	X
FMT_MSA.3.1/EXT_MEM	*EMG*			X	X

Table 27: Security Functional Requirements of Configurations

Finally, Table 28 summarizes the roles associated with each configuration:

Configuration	Roles
Minimal	Java Card RE, authorized role (CMGRG), Bytecode Verifier.
Java Card System Standard 2.1.1	Java Card RE, Installer, authorized role (CarG), authorized role (CMGRG), Bytecode Verifier.
Java Card System Standard 2.2	Java Card RE, Installer, authorized role (CarG), authorized role (CMGRG), applet deletion manager, applets (RMIG), Bytecode

	(CMGRG), applet deletion manager, applets (RMIG), Bytecode Verifier.
Defensive	Java Card RE, Installer, authorized role (CMGRG), applet deletion manager, applets (RMIG), Bytecode Verifier.

Table 28: Configurations and Roles

8 APPENDIX: GLOSSARY

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* **applet**s *and* *package*s. *AIDs* are administered by the International Standards Organization (ISO), so they can be used as unique identifiers.

AIDs are also used in the security policies (see "*Context*" below): applets' *AIDs* are related to the selection mechanisms, *package*s' *AIDs* are used in the enforcement of the *firewall*. **Note:** although they serve different purposes, they share the same name space.

APDU

Application Protocol Data Unit, an ISO 7816-4 defined communication format between the card and the off-card applications. Cards receive requests for service from the CAD in the form of *APDU*s. These are encapsulated in Java Card System by the **`javacard.framework.APDU`** *class* ([JCAPI21]).

*APDU*s 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 manager

The on-card component that embodies the mechanisms necessary to delete an applet or library and its associated data on smart cards using Java Card technology.

BCV

The bytecode verifier is the software component performing a static analysis of the code to be loaded on the card. It checks several kinds of properties, like the correct format of *CAP file*s and the enforcement of the typing rules associated to bytecodes. If the component is placed outside the card, in a secure environment, then it is called an off-card verifier. If the component is part of the embedded software of the card it is called an on-card verifier.

CAD — Card Acceptance Device, or card reader. The device where the card is inserted, and which is used to communicate with the card. Unless explicitely 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 *class*es that can be installed on a device and used to execute the *package*'s *class*es on a Java Card virtual machine. A CAP file can contain a user library, or the code of one or more applets.

Class — In object-oriented programming languages, a class is a prototype for an object. A class may also be considered as a set of objects that share a common structure and behavior. Each class declares a collection of fields and methods associated to its instances. The contents of the fields determine the internal state of a class instance, and the methods the operations that can be applied to it. Classes are ordered within a class hierarchy. A class declared as a specialization (a subclass) of another class (its super class) inherits all the fields and methods of the latter.

Java platform classes should not be confused with the classes of the functional requirements (FIA) defined in the CC.

Context — A context is an object-space partition associated to a *package*. Applets within the same Java technology-based *package* belong to the same context. The *firewall* is the boundary between contexts (see "*Current context*").

Current context — The *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 recieved over each interface to the currently selected applet on this interface ([JCRE21] , [JCRE222] Glossary).

Default applet — The applet that is selected after a card reset ([JCRE21], §4.1) or upon completion of the PICC activation sequence on the contactless interface ([JCRE222], §4.1)

Embedded Software — Pre-issuance loaded software.

Firewall — The mechanism in the Java Card technology for ensuring *applet* isolation and object sharing. The firewall prevents an applet in one *context* from unauthorized access to objects owned by the *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 *package*s (*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 ([JCRE21], §10).

Interface

A special kind of Java programming language *class*, which declares methods, but provides no implementation for them. A class may be declared as being the implementation of an interface, and in this case must contain an implementation for each of the methods declared by the interface (See also *shareable interface*).

Java Card RE

The Java Card runtime environment consists of the Java Card virtual machine, the Java Card API, and its associated native methods. This notion concerns all those dynamic features that are specific to the execution of a Java program in a smart card, like *applet* lifetime, applet isolation and object sharing, transient objects, the transaction mechanism, and so on.

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, mechanism enabling a client application running on the *CAD* platform to invoke a method on a remote object on the card. Notice that in Java Card System, version 2.1.1, the only method that may be invoked from the CAD is the **`process`** method of the **`applet`** class.

Java Card System

The Java Card System: the *Java Card RE* (*Java Card VM* +API), the *installer*, and the on-card *BCV* (if the configuration includes one).

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, 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, enables opening up to twenty logial channels over each I/O inteface (contacted or contactless).

Object deletion The Java Card System, version 2.2, mechanism ensures that any unreferenced persistent (transient) object owned by the current context is deleted. The associated memory space is recovered for reuse prior to the next card reset.

Package A *package* is a name space within the Java programming language that may contain *class*es and *interface*s. A *package* defines either a user library, or one or more applet definitions. A *package* is divided in two sets of files: export files (which exclusively contain the public *interface* information for an entire *package* of *class*es, for external linking purposes; export files are not used directly in a Java Card virtual machine) and *CAP file*s.

PCD Proximity Coupling Device. The PCD is a contactless card reader device.

PICC Proximity Card. The PICC is a card with contactless capabilities.

SCP Smart Card Platform. It is comprised of the integrated circuit, the operating system and the dedicated software of the smart card.

Shareable interface An interface declaring a collection of methods that an *applet* accepts to share with other applets. These *interface* methods can be invoked from an *applet* in a *context* different from the context of the object implementing the methods, thus "traversing" the *firewall*.

SIO An object of a class implementing a *shareable interface*.

Subject An active entity within the TOE that causes information to flow among objects or change the system's status. It usually acts on the behalf of a user. Objects can be active and thus are also *subjects* of the TOE.

Transient object An object whose contents is not preserved across CAD sessions. The contents of these objects are cleared at the end of the current CAD session or when a card reset is performed. Writes to the fields of a transient object are not affected by transactions.

User Any application interpretable by the *Java Card RE*. That also covers the *package*s. The associated subject(s), if applicable, is (are) an object(s) belonging to the **`javacard.framework.applet`** *class*.

End of Document