Java Card System – Open Configuration Protection Profile

July 2024
Version 3.2

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

Java Card System – Open Configuration Protection Profile
Version 3.2

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

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

- **Oracle Corporation,**
  2300 Oracle Way
  Austin, TX 78741,
  USA
  http://www.oracle.com
  seceval_us@oracle.com.

  Bundesamt für Sicherheit in der Informationstechnik
  Postfach 200363
  53133 Bonn, Germany
  https://www.bsi.bund.de/
  bsi@bsi.bund.de

Executive Summary

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

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

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

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

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

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

configuration is addressed in a dedicated Protection Profile conformant to Common Criteria version 2022.

The Closed and Open Configuration address versions 2.2.x and versions 3.x.x Classic Edition of Java Card Platform specifications. The Closed Configuration addresses Java Card Systems without post-issuance loading and installation of applets; the Open Configuration addresses Java Card Systems with full capabilities, in particular post-issuance content management; the Remote Method Invocation is optional. Both configurations consider off-card bytecode verification.

The Java Card System - Open Configuration Protection Profile version 3.2 replaces the Java Card System – Open Configuration Protection Profile, version 3.1, registered under the reference BSI-CC-PP-0099-V2-2020 (Bundesamt für Sicherheit in der Informationstechnik (BSI)).

TABLE OF CONTENTS

Figures

Tables

1 INTRODUCTION

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

1.1 PROTECTION PROFILE IDENTIFICATION

Title: Java Card System - Open Configuration Protection Profile

Version: 3.2

Publication date: 12th July 2024

Certified by: Bundesamt für Sicherheit in der Informationstechnik

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

Review Committee: Java Card Forum – Common Criteria Subgroup

This Protection Profile is conformant to the Common Criteria version 2022 revision 1.

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

1.2 PROTECTION PROFILE PRESENTATION

This Protection Profile replaces the Java Card System – Open Protection Profile 3.1 [PP JCS3.1]. It has been developed by Oracle Corporation with the aim of providing a full set of up-to-date security requirements: it relies on current Common Criteria version 2022 revision 1 and it addresses version 3 Classic Edition of the Java Card Specifications, namely [JCVM3], [JCAPI3] and [JCRE3]. Those specifications[1] cover the Java Card platform virtual machine ("Java Card virtual machine" or "Java Card VM"), the Java Card pla
tform runtime environment ("Java Card runtime environment" or "Java Card RE") and the Java Card Application Programming Interface (API).

This Protection Profile applies to evaluations of open Java Card Platforms, that is, smart cards or similar devices enabled with Java Card technology that support post-issuance downloading of applications, referred to as Java Card technology-based applets ("Java Card applets" or

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

"applets"). The Java Card technology combines a subset of the Java programming language with a runtime environment optimized for smart cards and similar small-memory embedded devices. The main security goal of the Java Card platform is to counter the unauthorized disclosure or modification of the code and data (keys, PINs, biometric templates, etc) of applications and platform. In order to achieve this goal, the Java Card System provides strong security features such as the secure installation mechanism, firewall mechanism, dedicated API for security services, etc.

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

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

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

Application note:

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

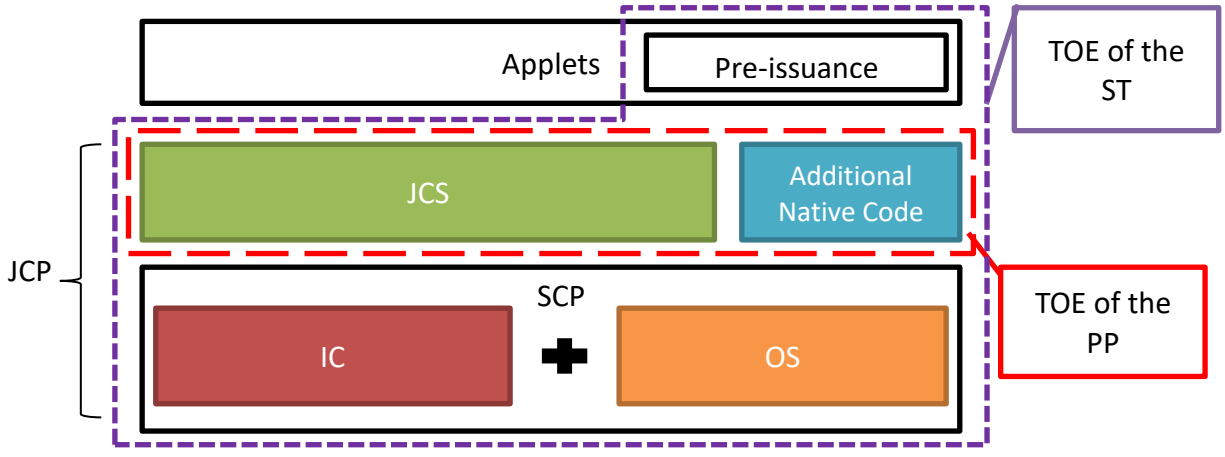

Figure 1: Java Card Platform

This Protection Profile focuses on the security requirements for the JCS and considers the SCP as the environment of the TOE, thus covered by security objectives. Nevertheless, any smart

card evaluation against[2] this PP shall comprehend the IC and all the embedded software, including the OS, the JCS, as well as the additional native code (if any) and the pre-issuance applets (if any). That is, the TOE of the ST conformant to this PP is as shown in Figure 1. The aim of introducing all the native code in the scope of the evaluation is to test that the native code that does not implement any security function cannot be used to circumvent or jeopardize the JCS TSFs.

This Protection Profile does not require formal compliance to a specific IC Protection Profile or a smart card OS Protection Profile but those IC and OS evaluated against [PP0084b] and [PP117] respectively, fully meet the objectives.

This Protection Profile requires "demonstrable" conformance.

The PP has been certified by the German Scheme Bundesamt für Sicherheit in der Informationstechnik.

The structure of this document is as follows:

- Chapter 2 presents an overview of the TOE, its security features and its life cycle.
- Chapter 3 defines the conformance claims applicable to this Protection Profile.
- Chapter 4 introduces general Java Card System security concerns, called "security aspects".
- Chapter 5 presents the assets of the JCS, the links between users and subjects, the relevant threats, the organisational security policies, and the assumptions.
- Chapter 6 describes the TOE security objectives, the security objectives for the operational environment and the security objectives rationale.
- Chapter 7 defines the TOE security functional and assurance requirements, the security requirements rationales, the dependencies analysis and the rationales for assurance requirements.
- Appendix 1 describes the augmentation packages introduced in Java Card System versions 2.2.2, 3.0.5 and 3.1 (External Memory, Biometric templates, SensitiveResult, Certificate API, Monotonic Counter API and Systems Time API). This appendix is only for informative purposes.
- Appendix 2 provides a comprehensive view of the two Java Card System Protection Profiles, Open and Closed Configurations.
- Appendix 3 contains a table of supported cryptographic algorithms.
- Appendix 4 contains a glossary of technical terms used in this document.

1.3 REFERENCES

[CC1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model. Version 2022. Revision 1. November 2022. CCMB-2022-11-001.

[CC2] Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components. Version 2022. Revision 1. November 2022. CCMB-2022-11-002.

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

[CC3] Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components. Version 2022. Revision 1. November 2022. CCMB-2022-11-003.

[CC4] Common Criteria for Information Technology Security Evaluation, Part 4: Framework for the specification of evaluation methods and activities. Version 2022. Revision 1. November 2022. CCMB-2022-04-004.

[CC5] Common Criteria for Information Technology Security Evaluation, Part 5: Pre-defined packages of security requirements. Version 2022. Revision 1. November 2022. CCMB-2022-11-005.

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology. Version 2022. Revision 1. November 2022. CCMB-2022-11-006.

[GP] GlobalPlatform Card Specification, Version 2.3.1, March 2018.

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

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

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

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

[JCAPI221] Java Card Platform, version 2.2.1 Application Programming Interface. October 2003. Published by Sun Microsystems, Inc.

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

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

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

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

[JCVM3] Java Card Platform, versions 3.0 up to 3.2, Classic Edition, Virtual Machine (Java Card VM) Specification. Published by Oracle.

[JCAPI3] Java Card Platform, versions 3.0 up to 3.2, Classic Edition, Application Programming Interface. Published by Oracle.

[JCRE3] Java Card Platform, versions 3.0 up to 3.2, Classic Edition, Runtime

Environment (Java Card RE) Specification. Published by Oracle.

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

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

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

[PP0084b] BSI-CC-PP-0084-2014, Security IC Platform Protection Profile with Augmentation Packages, Version 1.0, 13 January 2014

[PP117] BSI-CC-PP-0117-V2-2023, Secure Sub-System in System-on-Chip (3S in SoC) Protection Profile, Version 1.8, 26 October 2023

[PP JCS3.1] Java Card System Protection Profile, Version 3.1, april 2020, registered and certified by the German certification body (BSI) under the following references: [BSI-CC-PP-0099-V2-2020].

2 TOE OVERVIEW

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

2.1 TOE TYPE

2.1.1 TOE OF THIS PP

The TOE type in this PP is the Java Card System (Java Card RE, Java Card VM and Java Card API) along with the additional native code embedded in a Smart Card Platform. The Java Card System is compliant with Java Card specifications versions 2.2.x or 3.x.x Classic Edition, including post-issuance installation facilities of applications verified off-card. Native code post-issuance downloading is out of the scope in this PP.

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

2.1.2 TOE OF THE ST

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

Application note:

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

2.2 TOE SECURITY FUNCTIONS

The Java Card System Open Configuration considered in this Protection Profile implements Java Card Specifications versions 2.2.x or 3.x.x Classic Edition and allows post-issuance downloading of applications that have been previously verified by an off-card trusted IT component.
Figure 2 shows the Java Card System and the relationship with the environment for applet installation purposes in two scenarios: one relying on off-card verification (black lines),

described hereafter, the other one relying on on-card verification by the installer (dotted red lines).

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

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

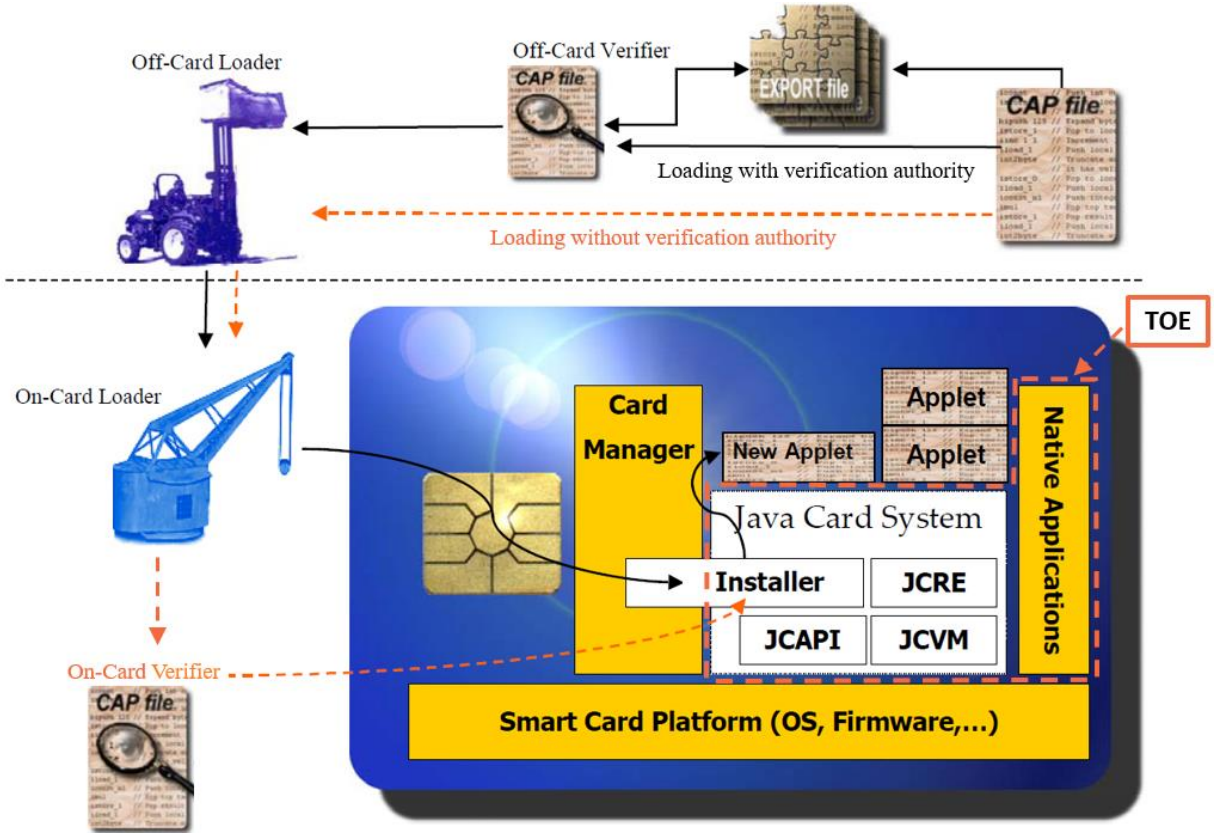

Figure 2: Java Card System and applet installation environment

[3] The converter is defined in the specifications [JCVM3] as the off-card component of the Java Card virtual machine.

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

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

The installer is the Java Card System component dealing with loading, linking and installation of new CAP files, as described in [JCRE22]. Once selected, it receives the CAP file, stores the classes of the CAP file on the card, initializes static data, if any, and installs any applets contained in the CAP file. The installer is also in charge of applet deletion ([JCRE3], §11.3.4):

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

The Java Card VM is the bytecode interpreter as specified in [JCVM3]. The Java Card RE is responsible for card resource management, communication, applet execution, on-card system and applet security. The Java Card API provides classes and interfaces to the Java Card applets. It defines the calling conventions by which an applet may access the Java Card RE and native services such as, I/O management functions, PIN and cryptographic specific management and the exceptions mechanism.

While the Java Card VM is responsible for ensuring language-level security, the Java Card RE provides additional security features for Java Card technology-enabled devices. Applets from different vendors can coexist in a single card, and they can even share information. An applet, however, is usually intended to store highly sensitive information, so the sharing of that information must be carefully limited. In the Java Card platform, applet isolation is achieved through the applet firewall mechanism ([JCRE3] §6.1). That mechanism confines an applet to its own designated memory area, thus each applet is prevented from accessing fields and operations of objects owned by other applets, unless a "shareable interface" is explicitly provided ~~(~~by the applet who owns it~~)~~ for allowing access to that information. The Java Card RE allows sharing using the concept of "shareable interface objects" (SIO) and static public variables. Java Card VM dynamically enforces the firewall, that is, at runtime. However applet

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

isolation cannot be entirely granted by the firewall mechanism if certain integrity conditions are not satisfied by the applications loaded on the card. Those conditions can be statically verified to hold by a bytecode verifier ([JCRE3], §6.1.1).

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

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

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

Array Views, API Extensibility and Static Resources are introduced as mandatory features in version 3.1 of Java Card Specifications. Java Card systems that are compliant with previous versions of Java Card Specifications may not provide these features. Therefore, the TOE shall only provide these features when the Java Card System is compliant with version 3.1.x and the following versions (e.g 3.2) of Java Card Specifications.

Array Views, introduced in version 3.1 of Java Card Specifications allow creation of a "view" on a subset of elements of an existing array. The elements of the view are mapped to the elements of the actual array to avoid defensive copies, synchronization protocols, and to allow a fine-grained access control on the elements.

API Extensibility, introduced in version 3.1 of Java Card Specifications, removes the CAP file limitation that prevented adding methods to the non-final classes. Additional information is added to the CAP file structure for an enhanced virtual method token assignment, which allows evolution of the platform API or shared libraries without breaking binary compatibility rules.

Static Resources in CAP files, also introduced in version 3.1 of Java Card Specifications, allow an application to embed static resources such as configuration data or initialization data, in the CAP file and access these resources from the application code.

Logical channels allow a terminal to open multiple sessions into the smart card, one session per logical channel ([JCRE3], §4). Commands may be issued on a logical channel to instruct the card either to open or to close a logical channel. An applet instance that is selected to be active on a channel shall process all the commands issued to that channel. The platform also introduces the possibility for an applet instance to be selected on multiple logical channels at the same time, or accepting other applets belonging to the same CAP file to be selected simultaneously. These applets are referred to as multiselectable. A non-multiselectable applet can be active at most on one channel. Applets within a CAP file are either all multiselectable or all non-multiselectable.

Support of Logical Channel CLA byte encoding is mandatory in versions 2.2.x up to 3.1.x versions of Java Card specification. Starting version 3.2, it is optional. The JCS developer may choose not to activate this functionality in the TOE and may support only the basic logical channel

(logical channel 0) on its I/O interfaces. For these reasons, this PP considers that the TOE may or may not provide the Logical Channel functionality.

In addition, the Java Card System compliant with Java Card version 3.1 and subsequent versions (e.g 3.2) must support:

- API extensibility using virtual method tables.
- Array view object types. Array view objects are type of arrays with elements mapped to the elements of another array.
- Static Resources in CAP files

The Java Card System may optionally provide:

- Object deletion upon request of an applet instance. The JCRE ensures that any unreferenced object owned by that instance is deleted and the associated space is recovered for reuse.
- Extended memory facilities, introduced in Java Card specification version 2.2.2. This is an API-based mechanism to access the external memory outside the addressable Java Card VM space.
- Support for Biometric Template management, introduced in Java Card Specification 2.2.2. This feature is described by the augmentation package in Appendix 1.
- SensitiveResult, introduced in Java Card specification version 3.0.5. This API provides methods to assert results of sensitive functions. This feature is described by the augmentation package in Appendix 1.
- SensitiveArray, introduced in Java Card specification version 3.0.5, allow creating and handling of integrity-sensitive array objects. This feature is described by the augmentation package in Appendix 1.

The Java Card System compliant with Java Card specification version 3.1 and the following versions (e.g 3.2) may also optionally provide:

- Support for Extended CAP files introduced in Java Card specification version 3.1. Extended CAP files may contain multiple Java language packages. An Extended CAP file may contain only applet packages, only library packages or a combination of library and applet packages.
- Implementation for Pseudo Random Functions (PRF) and Key Derivation Function (KDF) APIs (See augmentation package in Appendix 1).
- Implementation for Certificate APIs.
- Implementation for Monotonic Counter API (See augmentation package in Appendix 1).
- Implementation for System Time API (See augmentation package in Appendix 1).

Java Card System 3 Classic Edition provides support for ETSI defined SWP protocol for contactless communication, and for independent contacted and contactless interfaces. Moreover, it provides support for USB connected interface communication. In version 3.1 of the Java Card specifications, API has been enhanced by introducing an abstract application layer and thereby allowing support for many physical I/O layers.

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

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

2.3.1 CAP FILE CONVERSION AND EXPORT FILES

Java Card technology uses a binary representation of programs based on two separate files CAP files (for converted applet) and export files as described in [JCVM3].

The Converter is a piece of software that preprocesses all of the Java programming language class files contained in a set of packages and converts them into a CAP file. The Converter also produces *export files* for exported packages.

A Java Card CAP file contains a binary representation of a Java Card application or library or both, consisting of one or more packages of classes that can be installed on a device and used to execute the Java Card application or library's classes on a Java Card virtual machine.
A CAP file is produced by a Java Card converter when a Java Card application or library is converted. A CAP file consists of a set of components, each of which describes a different aspect of the contents. The set of components in a CAP file can vary, depending on whether the file contains a library or applet definition(s). Since Java Card 3.1, a CAP file can be in Compact or Extended format where a CAP file in Compact format can only contain a single Java package and a CAP file in Extended format may contain one or more packages.
The actual set of validation of class files performed by the converter is implementation-dependent but it is required that it generates CAP files as specified in [JCVM3]. It is required also that the latest available version of the converter is used.

Export files are not used directly on a device that implements a Java Card virtual machine. However, the information in an export file is critical to the operation of the virtual machine on a device. An export file can be produced by a Java Card converter when a package is converted. This package's export file can be used later to convert another package that imports classes from the first package. Information in the export file is included in the CAP file of the second package, then is used on the device to link the contents of the second package to items imported from the first package.
When export files are generated by the converter, it is recommended that the latest version of the export file format specified in [JCVM3] is used (i.e. export file format version 2.3 for the Java Card Virtual Machine Version 3.2).

2.3.2 BYTECODE VERIFICATION

The bytecode verifier is a program that performs static checks on the bytecodes of the methods of a CAP file prior to the execution of the file on the card. Bytecode verification is a key component of security: applet isolation, for instance, depends on the file satisfying the properties a verifier checks to hold. A method of a CAP file that has been verified shall not contain, for instance, an instruction that allows forging a memory address or an instruction that makes improper use of a return address as if it were an object reference. In other words, bytecodes are verified to hold up to the intended use to which they are defined. Bytecode verification could be performed totally or partially dynamically. No standard procedure in that

concern has yet been recognized. Furthermore, different approaches have been proposed for the implementation of bytecode verifiers, most notably data flow analysis, model checking and lightweight bytecode verification, this latter being an instance of what is known as proof carrying code. The actual set of checks performed by the verifier is implementation-dependent, but it is required that it should at least enforce all the "must clauses" imposed in [JCVM22] on the bytecodes and the correctness of the CAP files' format. It is required also that the latest available version of the verifier is used.

2.3.3 THE CARD MANAGER (CM)

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

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

2.3.4 SMART CARD PLATFORM

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

2.4 TOE LIFE CYCLE

The Java Card System (the TOE) life cycle is part of the product life cycle, i.e. the Java Card platform with applications, which goes from product development to its usage by the final user. The product life cycle phases are those detailed in Figure 3. We refer to [PP0084b] for a thorough description of Phases 1 to 7:

- Phases 1 and 2 compose the product development: Embedded Software (IC Dedicated Software, OS, Java Card System, other platform components such as Card Manager, Applets) and IC development.
- Phase 3 and Phase 4 correspond to IC manufacturing and packaging, respectively. Some IC pre-personalization steps may occur in Phase 3.

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

- Phase 5 concerns the embedding of software components within the IC.
- Phase 6 is dedicated to the product personalization prior final use.
- Phase 7 is the product operational phase.

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

- Development,
- Storage, pre-personalization and testing
- Personalization and testing
- Final usage.

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

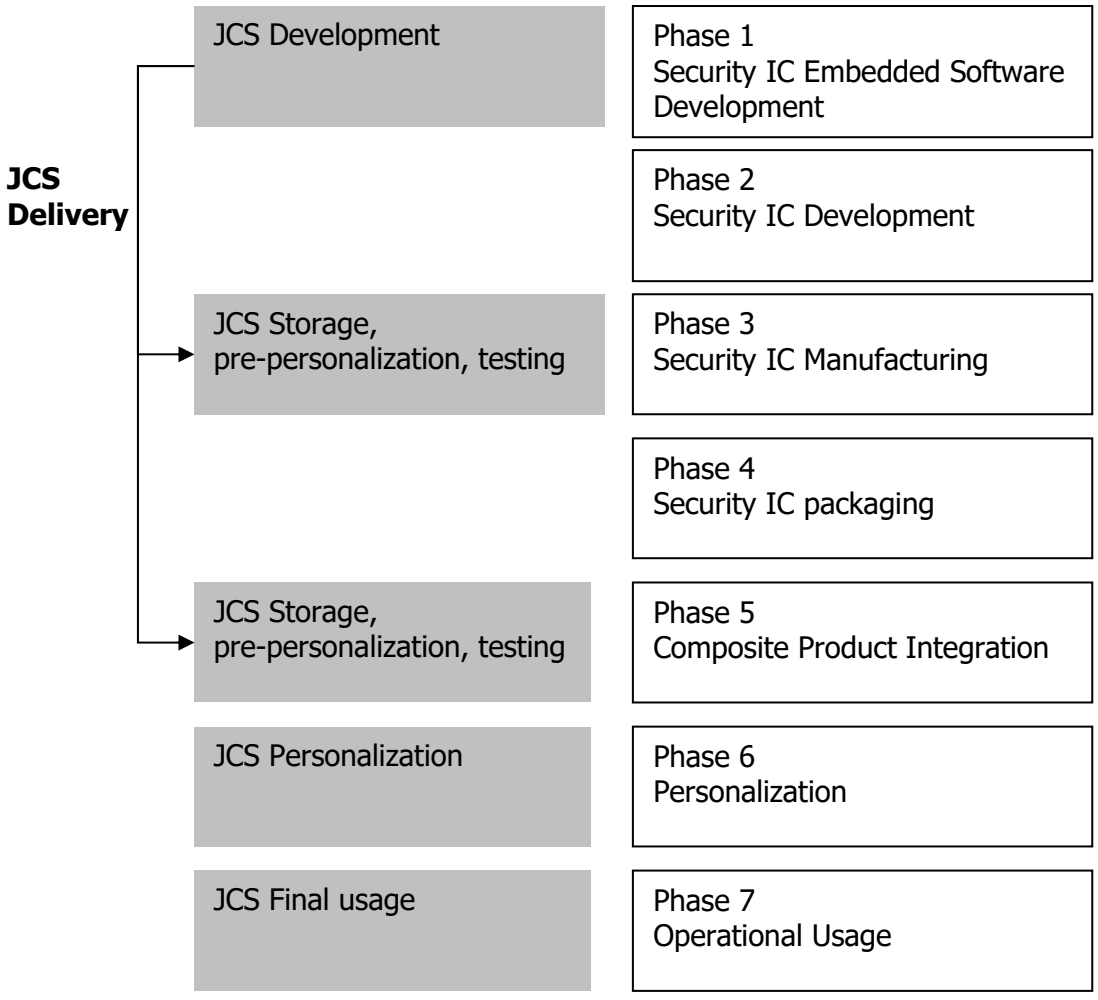

Figure 3: JCS (TOE) Life Cycle within Product Life Cycle

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

The delivery of the JCS may occur either during Security IC Manufacturing (Phase 3) or during Composite Product Integration (Phase 5). It is also possible that part of the JCS is delivered in Phase 3 and the rest is delivered in Phase 5. Delivery and acceptance procedures shall guarantee the authenticity, the confidentiality and integrity of the exchanged pieces. JCS delivery shall usually involve encrypted signed sending and it supposes the previous exchange of public keys. The evaluation of a product against this PP shall include the delivery process.

In Phase 3, the Security IC Manufacturer may store, pre-personalize the JCS and potentially conduct tests on behalf of the JCS developer. The Security IC Manufacturing environment shall protect the integrity and confidentiality of the JCS and of any related material, for instance test suites. The evaluation of a product against this PP shall include the whole Security IC Manufacturing environment, in particular those locations where the JCS is accessible for installation or testing. If the Security IC has already been certified (e.g. against [PP0084b] or [PP117]) there is no need to perform the evaluation again.

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

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

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

Application note:

The Security Target writer shall specify the life cycle of the product, the JCS delivery point and the product delivery point. The product delivery point may arise at the end of Phase 3, 4, 5 or

6 depending on the product itself. Note that JCS delivery precedes product delivery. During product evaluation against this Protection Profile, the ALC security assurance requirements apply to the whole product life cycle up to delivery.

2.5 TOE USAGE

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

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

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

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

So far, the most typical applications are:

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

3 CONFORMANCE CLAIMS

3.1 CC CONFORMANCE CLAIMS

This Protection Profile is CC Part 2 [CC2] and CC Part 3 [CC3] conformant of Common Criteria version 2022, revision 1. This PP is CC Part 4 [CC4] conformant. However, CC Part 4 is not used in this PP.

3.2 CONFORMANCE CLAIM TO A PACKAGE

The minimum assurance level for the evaluation of a Java Card Platform with a TOE conformant to this PP is EAL4 augmented with AVA_VAN.5 "Advanced methodical vulnerability analysis", ALC_DVS.2 "Sufficiency of security measures" and ALC_FLR.2 "Flaw reporting procedures". This package is CC Part 5 [CC5] conformant.

3.3 PROTECTION PROFILE CONFORMANCE CLAIMS

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

3.4 CONFORMANCE CLAIMS TO THIS PROTECTION PROFILE

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

4 SECURITY ASPECTS

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

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

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

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

4.1 CONFIDENTIALITY

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

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

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

4.2 INTEGRITY

#.INTEG-APPLI-CODE Application code must be protected against unauthorized modification. This concerns logical attacks at runtime in order to gain write access to the memory zone where executable code is stored. In 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. In post-issuance application loading, this threat also concerns the modification of application data contained in a CAP file in transit to the card. For instance, a CAP file contains the values to be used for initializing the static fields of the CAP file.

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

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

4.3 UNAUTHORIZED EXECUTIONS

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

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

#.FIREWALL The Firewall shall ensure controlled sharing of class instances[7], and isolation of their data and code between CAP files (that is, controlled

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

execution contexts) as well as between CAP files and the JCRE context. An applet shall not read, write, compare a piece of data belonging to an applet that is not in the same context, or execute one of the methods of an applet in another context without its authorization.

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

4.4 BYTECODE VERIFICATION

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

4.4.1 CAP FILE VERIFICATION

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

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

Another important aspect to be considered about bytecode verification and application downloading is, first, the assurance that every package required by the loaded applet is indeed on the card, in a binary-compatible version (binary compatibility is explained in [JCVM22] §4.4),

second, that the export files used to check and link the loaded applet have the corresponding correct counterpart on the card.

4.4.2 INTEGRITY AND AUTHENTICATION

Verification off-card is useless if the application CAP file 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 CAP file verification and CAP file installation. Once a verification authority has verified the CAP file, it signs it and sends it to the card. Prior to the installation of the CAP file, the card verifies the signature of the CAP file, which authenticates the fact that it has been successfully verified. In addition to this, a secured communication channel is used to communicate it to the card, ensuring that no modification has been performed on it.

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

A verification authority may serve different applet providers in which case a mechanism must enforce the verification, integrity and authentication of a CAP file on behalf of the verification authority. For instance, the Mandated DAP (Data Authentication Pattern) Verification privilege of the Verification Authority's Security Domain defined within GlobalPlatform specifications [GP] provides such mechanism.

4.4.3 LINKING AND VERIFICATION

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

4.5 CARD MANAGEMENT

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

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

#.SID (1) Users and subjects of the TOE must be identified. (2) The identity of sensitive users and subjects associated with administrative and privileged roles must be particularly protected; this concerns the Java Card RE, the applets registered on the card, and especially the default applet and the currently selected applet (and all other active applets in Java Card System 2.2.x). A change of identity, especially standing for an administrative role (

like an applet impersonating the Java Card RE), is a severe violation of the Security Functional Requirements (SFR). Selection controls the access to any data exchange between the TOE and the CAD and therefore, must be protected as well. The loading of a CAP file 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 applets (or CAP files) should not introduce security holes in the form of broken references to garbage collected code or data, nor should they alter integrity or confidentiality of remaining applets. The deletion procedure should not be maliciously used to bypass the TSFs. (2) Erasure, if deemed successful, shall ensure that any data owned by the deleted applet is no longer accessible (shared objects shall either prevent deletion or be made inaccessible). A deleted applet cannot be selected or receive APDU commands. CAP file deletion shall make the code of the CAP file is no longer available for execution. In case of Extended CAP files, deletion of a CAP shall ensure that code and data for all the packages in the CAP file is no longer available for execution. (3) Power failure or other failures during the process shall be taken into account in the implementation so as to preserve the SFRs. This does not mandate, however, the process to be atomic. For instance, an interrupted deletion may result in the loss of user data, as long as it does not violate the SFRs.

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

Deletion of a single applet instance and deletion of a whole CAP file are functionally different operations and may obey different security rules. For instance, specific CAP files or packages can be declared to be undeletable (for instance, the Java Card API packages), or the dependency between installed CAP files may forbid the deletion (like a

CAP file using super classes or super interfaces declared in another CAP file).

4.6 SERVICES

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

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

#.RESOURCES The TOE controls the availability of resources for the applications 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 CAP files.

#.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) Protection of PIN's security attributes (state, try counter, try limit, …) in integrity.

#.SCP The smart card platform must be secure with respect to the SFRs. Then: (1) After a power loss, RF signal loss or sudden card removal prior to completion of some communication protocol, the SCP will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state. (2) It does not allow the SFRs to be bypassed or altered and does not allow access to other low-level functions than those made available by packages of Java Card API. That includes the protection of its private data and code (against disclosure or modification) from the Java Card System. (3) It provides secure low-level cryptographic processing to the Java Card System.

(4) It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism. (5) It allows the Java Card System to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low–level control accesses (segmentation fault detection). (6) It safely transmits low–level exceptions to the TOE (arithmetic exceptions, checksum errors), when applicable. Finally, it is required that (7) the IC is designed in accordance with a well-defined set of policies and standards (for instance, those specified in [PP0084b] or [PP117]), 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 confidential application data such as cryptographic keys.

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

5 SECURITY PROBLEM DEFINITION

5.1 ASSETS

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

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

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

5.1.1 USER DATA

D.APP_CODE

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

To be protected from unauthorized modification.

D.APP_C_DATA

Confidentiality - sensitive data of the applications, like the data contained in an object, an array view, a static field, 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, an array view and the PIN security attributes (PIN Try limit, PIN Try counter and State).

To be protected from unauthorized modification.

D.APP_KEYs

Cryptographic keys owned by the applets.

To be protected from unauthorized disclosure and modification.

D.PIN

Any end-user's PIN.

To be protected from unauthorized disclosure and modification.

5.1.2 TSF DATA

D.API_DATA

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

To be protected from unauthorized disclosure and modification.

D.CRYPTO

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

To be protected from unauthorized disclosure and modification.

D.JCS_CODE

The code of the Java Card System.

To be protected from unauthorized disclosure and modification.

D.JCS_DATA

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

To be protected from unauthorized disclosure or modification.

D.SEC_DATA

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

To be protected from unauthorized disclosure and modification.

5.2 THREATS

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

5.2.1 CONFIDENTIALITY

T.CONFID-APPLI-DATA

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

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

T.CONFID-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.CONFID-JCS-DATA

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

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

5.2.2 INTEGRITY

T.INTEG-APPLI-CODE

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-CODE.LOAD

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-DATA

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

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

T.INTEG-APPLI-DATA.LOAD

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

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

T.INTEG-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.INTEG-JCS-DATA

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

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

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

5.2.3 IDENTITY USURPATION

T.SID.1

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

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

T.SID.2

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

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

5.2.4 UNAUTHORIZED EXECUTION

T.EXE-CODE.1

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

Directly threatened asset(s): D.APP_CODE.

T.EXE-CODE.2

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

Directly threatened asset(s): D.APP_CODE.

T.NATIVE

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

Directly threatened asset(s): D.JCS_DATA.

5.2.5 DENIAL OF SERVICE

T.RESOURCES

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

Directly threatened asset(s): D.JCS_DATA.

5.2.6 CARD MANAGEMENT

T.DELETION

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

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

T.INSTALL

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

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

5.2.7 SERVICES

T.OBJ-DELETION

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

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

5.2.8 MISCELLANEOUS

T.PHYSICAL

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

This threatens all the identified assets.

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

5.3 ORGANISATIONAL SECURITY POLICIES

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

OSP.VERIFICATION

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

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

5.4 ASSUMPTIONS

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

A.CAP_FILE

CAP Files loaded post-issuance do not contain native methods. The Java Card specification explicitly "does not include support for native methods" ([JCVM3], §3.3) outside the API.

A.DELETION

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

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, to ensure that each bytecode is valid at execution time. The latest available version of the verifier is used.

6 SECURITY OBJECTIVES

6.1 SECURITY OBJECTIVES FOR THE TOE

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

6.1.1 IDENTIFICATION

O.SID

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

6.1.2 EXECUTION

O.FIREWALL

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

O.GLOBAL_ARRAYS_CONFID

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

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

O.GLOBAL_ARRAYS_INTEG

The TOE shall ensure that no application can store a reference to the APDU buffer, a global byte array created by the user through makeGlobalArray method and the byte array used for invocation of the install method of the selected applet.

O.ARRAY_VIEWS_CONFID

The TOE shall ensure that no application can read elements of an array view not having array view security attribute ATTR_READABLE_VIEW.

The TOE shall ensure that an application can only read the elements of the array view within the bounds of the array view.

O. ARRAY_VIEWS_INTEG

The TOE shall ensure that no application can write to an array view not having array view security attribute ATTR_WRITABLE_VIEW.

The TOE shall ensure that an application can only write within the bounds of the array view.

O.NATIVE

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

O.OPERATE

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

O.REALLOCATION

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

O.RESOURCES

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

6.1.3 SERVICES

O.ALARM

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

O.CIPHER

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

O.RNG

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

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

O.KEY-MNGT

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

O.PIN-MNGT

The TOE shall provide a means to securely manage PIN objects (including the PIN try limit, PIN try counter and states). If the PIN try limit is reached, no further PIN authentication must be allowed.

See #.PIN-MNGT for details.

Application Note:

PIN objects may play key roles in the security architecture of client applications. The way they are stored and managed in the memory of the smart card must be carefully considered,

and this applies to the whole object rather than the sole value of the PIN. For instance, the try limit and the try counter's value are as sensitive as that of the PIN and the TOE must restrict their modification only to authorized applications such as the card manager.

O.TRANSACTION

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

O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION, O.RNG and O.CIPHER are actually provided to applets in the form of Java Card APIs. Vendor-specific libraries can also be present on the card and made available to applets; those may be built on top of the Java Card API or independently. These proprietary libraries will be evaluated together with the TOE.

6.1.4 OBJECT DELETION

O.OBJ-DELETION

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

6.1.5 APPLET MANAGEMENT

O.DELETION

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

O.LOAD

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

Besides, for code loaded post-issuance, the TOE shall verify the integrity and authenticity evidences generated during the verification of the application CAP file by the verification authority. This verification by the TOE shall occur during the loading or later during the install process.

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 CAP files sent to the card. He may also try to send one of its own applications as if it came from the card issuer. Thus, this objective is intended to ensure the integrity and authenticity of loaded CAP files.

O.INSTALL

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

Besides, for code loaded post-issuance, the TOE shall verify the integrity and authenticity evidences generated during the verification of the application CAP file by the verification authority. If not performed during the loading process, this verification by the TOE shall occur during the install process.

6.2 SECURITY OBJECTIVES FOR THE OPERATIONAL ENVIRONMENT

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

OE.CAP_FILE

No CAP file loaded post-issuance shall contain native methods.

OE.CARD-MANAGEMENT

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

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

OE.SCP.IC

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

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

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

OE.SCP.RECOVERY

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

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

OE.SCP.SUPPORT

The SCP shall support the TSFs of the TOE.

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

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

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

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

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

OE.VERIFICATION

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

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

Application Note:

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

OE.CODE-EVIDENCE

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

For application code loaded post-issuance and verified off-card according to the requirements of OE.VERIFICATION, the verification authority shall provide digital evidence to the TOE that the application code has not been modified after the code verification and that he is the actor who performed code verification.

For application code loaded post-issuance and partially or entirely verified on-card, technical measures must ensure that the verification required in OE.VERIFICATION are performed. On-card bytecode verifier is out of the scope of this Protection Profile.

Application Note:

For application code loaded post-issuance and verified off-card, the integrity and authenticity evidence can be achieved by electronic signature of the application code, after code verification, by the actor who performed verification.

6.3 SECURITY OBJECTIVES RATIONALE

6.3.1 THREATS

6.3.1.1 CONFIDENTIALITY

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

isolation commitments stated in the (O.FIREWALL) objective. It relies in its turn on the correct identification of applets stated in (O.SID). Moreover, as the firewall is dynamically enforced, it shall never stop operating, as stated in the (O.OPERATE) objective.

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

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

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

As applets may need to share some data or communicate with the CAD, cryptographic functions are required to actually protect the exchanged information (O.CIPHER, O.RNG). Remark that even if the TOE shall provide access to the appropriate TSFs, it is still the responsibility of the applets to use them. Keys, PIN's are particular cases of an application's sensitive data (the Java Card System may possess keys as well) that ask for appropriate management (O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION). If the PIN class of the Java Card API is used, the objective (O.FIREWALL) shall contribute in covering this threat by controlling the sharing of the global PIN between the applets.

Other application data that is sent to the applet as clear text arrives to the APDU buffer, which is a resource shared by all applications. The disclosure of such data is prevented by the security objective O.GLOBAL_ARRAYS_CONFID.

An applet might share data buffer with another applet using array views without the array view security attribute ATTR_READABLE_VIEW. The disclosure of data of the applet creating the array view is prevented by the security object O.ARRAY_VIEWS_CONFID.

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

T.CONFID-JCS-CODE This threat is countered by the list of properties described in the (#.VERIFICATION) security aspect. Bytecode verification ensures that each of the instructions used on the Java Card platform is used for its intended purpose and in the intended scope of accessibility. As none of those instructions enables reading a piece of code, no Java Card applet can therefore be executed to disclose a piece of code. Native applications are also harmless because of the objective O.NATIVE, so no application can be run to disclose a piece of code.

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

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

T.CONFID-JCS-DATA This threat is covered by bytecode verification (OE.VERIFICATION) and the isolation commitments stated in the (O.FIREWALL) security objective. This latter objective also relies in its turn on the correct identification of applets stated in (O.SID).

Moreover, as the firewall is dynamically enforced, it shall never stop operating, as stated in the (O.OPERATE) objective.

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

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

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

6.3.1.2 INTEGRITY

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

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

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

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

T.INTEG-APPLI-CODE.LOAD This threat is countered by the security objective O.LOAD which ensures that the loading of CAP files is done securely and thus preserves the integrity of CAP files' code.

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

T.INTEG-APPLI-DATA This threat is countered by bytecode verification (OE.VERIFICATION) and the isolation commitments stated in the (O.FIREWALL) objective. This latter objective also relies in its turn on the correct identification of applets stated in (O.SID). Moreover, as the firewall is dynamically enforced, it shall never stop operating, as stated in the (O.OPERATE) objective.

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

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

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity. The objectives OE.SCP.RECOVERY and OE.SCP.SUPPORT are intended to support the O.OPERATE and O.ALARM objectives of the TOE, so they are indirectly related to the threats that these latter objectives contribute to counter.

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

Other application data that is sent to the applet as clear text arrives to the APDU buffer, which is a resource shared by all applications. The integrity of the information stored in that buffer is ensured by the objective O.GLOBAL_ARRAYS_INTEG.

An applet might share data buffer with another applet using array views without the array view security attribute ATTR_WRITABLE_VIEW. The integrity of data of the applet creating the array view is ensured by the security objective O.ARRAY_VIEWS_INTEG.

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

T.INTEG-APPLI-DATA.LOAD This threat is countered by the security objective O.LOAD which ensures that the loading of CAP files is done securely and thus preserves the integrity of applications data.

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

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

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

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

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

T.INTEG-JCS-DATA This threat is countered by bytecode verification (OE.VERIFICATION) and the isolation commitments stated in the (O.FIREWALL) objective. This latter objective also relies in its turn on the correct identification of applets stated in (O.SID). Moreover, as the firewall is dynamically enforced, it shall never stop operating, as stated in the (O.OPERATE) objective.

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

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

The objective OE.CODE-EVIDENCE contributes to cover this threat by ensuring that the application code loaded into the platform has not been changed after code verification, which ensures code integrity and authenticity. The objectives OE.SCP.RECOVERY and OE.SCP.SUPPORT are intended to support the O.OPERATE and O.ALARM objectives of the TOE, so they are indirectly related to the threats that these latter objectives contribute to counter.

6.3.1.3 IDENTITY USURPATION

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

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 objectives O.GLOBAL_ARRAYS_CONFID and O.GLOBAL_ARRAYS_INTEG.

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

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

The objective O.INSTALL contributes to counter this threat by ensuring that installing an applet has no effect on the state of other applets and thus can't change the TOE's attribution of privileged roles.

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

6.3.1.4 UNAUTHORIZED EXECUTION

T.EXE-CODE.1 Unauthorized execution of a method is prevented by the objective OE.VERIFICATION. This threat particularly concerns the point (8) of the security aspect #.VERIFICATION (access modifiers and scope of accessibility for classes, fields and methods). The O.FIREWALL objective is also concerned, because it prevents the execution

of non-shareable methods of a class instance by any subject apart from the class instance owner.

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

T.NATIVE This threat is countered by O.NATIVE which ensures that a Java Card applet can only access native methods indirectly that is, through an API. OE.**CAP_FILE** also covers this threat by ensuring that no CAP files containing native code shall be loaded in post-issuance. In addition to this, the bytecode verifier also prevents the program counter of an applet to jump into a piece of native code by confining the control flow to the currently executed method (OE.VERIFICATION).

6.3.1.5 DENIAL OF SERVICE

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

Consumption of resources during installation and other card management operations are covered, in case of failure, by O.INSTALL.

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

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

6.3.1.6 CARD MANAGEMENT

T.DELETION This threat is covered by the O.DELETION security objective which ensures that both applet and CAP file deletion perform as expected.

The objective OE.CARD-MANAGEMENT controls the access to card management functions and thus contributes to cover this threat.

T.INSTALL This threat is covered by the security objective O.INSTALL which ensures that the installation of an applet performs as expected and the security objectives O.LOAD which ensures that the loading of a CAP file into the card is safe.

The objective OE.CARD-MANAGEMENT controls the access to card management functions and thus contributes to cover this threat.

6.3.1.7 SERVICES

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

6.3.1.8 MISCELLANEOUS

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

6.3.2 ORGANISATIONAL SECURITY POLICIES

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

This policy is also upheld by the security objective of the environment OE.CODE-EVIDENCE which ensures that evidences exist that the application code has been verified and not changed after verification, and by the security objective for the TOE O.LOAD which shall ensure that the loading of a CAP file into the card is safe.

6.3.3 ASSUMPTIONS

A.CAP_FILE This assumption is upheld by the security objective for the operational environment OE.**CAP_FILE** which ensures that no CAP file loaded post-issuance shall contain native methods.

A.DELETION The assumption A.DELETION is upheld by the environmental objective OE.CARD-MANAGEMENT which controls the access to card management functions such as deletion of applets.

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

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

6.3.4 SPD AND SECURITY OBJECTIVES

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

T.NATIVE	OE.VERIFICATION, OE.CAP_FILE , O.NATIVE	Section 6.3.1
T.RESOURCES	O.INSTALL, O.OPERATE, O.RESOURCES, OE.SCP.RECOVERY, OE.SCP.SUPPORT	Section 6.3.1
T.DELETION	O.DELETION, OE.CARD-MANAGEMENT	Section 6.3.1
T.INSTALL	O.INSTALL, O.LOAD, OE.CARD-MANAGEMENT	Section 6.3.1
T.OBJ-DELETION	O.OBJ-DELETION	Section 6.3.1
T.PHYSICAL	OE.SCP.IC	Section 6.3.1

Table 1 Threats and Security Objectives - Coverage

Security Objectives	Threats
O.SID	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.1, T.SID.2
O.FIREWALL	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.1, T.SID.2, T.EXE-CODE.1
O.GLOBAL_ARRAYS_CONFID	T.CONFID-APPLI-DATA, T.SID.1
O.GLOBAL_ARRAYS_INTEG	T.INTEG-APPLI-DATA, T.SID.1
O.ARRAY_VIEWS_CONFID	T.CONFID-APPLI-DATA
O.ARRAY_VIEWS_INTEG	T.INTEG-APPLI-DATA
O.NATIVE	T.CONFID-JCS-CODE, T.INTEG-APPLI-CODE, T.INTEG-JCS-CODE, T.NATIVE
O.OPERATE	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
O.REALLOCATION	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.RESOURCES	T.RESOURCES
O.ALARM	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA
O.CIPHER	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.RNG	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA

O.KEY-MNGT	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.PIN-MNGT	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.TRANSACTION	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA
O.OBJ-DELETION	T.OBJ-DELETION
O.DELETION	T.DELETION
O.LOAD	T.INTEG-APPLI-CODE.LOAD, T.INTEG-APPLI-DATA.LOAD, T.INSTALL
O.INSTALL	T.SID.1, T.SID.2, T.RESOURCES, T.INSTALL
OE.CAP_FILE	T.NATIVE
OE.CARD-MANAGEMENT	T.CONFID-APPLI-DATA, T.CONFID-JCS-CODE, T.CONFID-JCS-DATA, T.INTEG-APPLI-CODE, T.INTEG-APPLI-CODE.LOAD, T.INTEG-APPLI-DATA, T.INTEG-APPLI-DATA.LOAD, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA, T.SID.1, T.DELETION, T.INSTALL
OE.SCP.IC	T.PHYSICAL
OE.SCP.RECOVERY	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
OE.SCP.SUPPORT	T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA, T.SID.2, T.RESOURCES
OE.VERIFICATION	T.CONFID-APPLI-DATA, T.CONFID-JCS-CODE, T.CONFID-JCS-DATA, T.INTEG-APPLI-CODE, T.INTEG-APPLI-DATA, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA, T.EXE-CODE.1, T.EXE-CODE.2, T.NATIVE
OE.CODE-EVIDENCE	T.INTEG-APPLI-CODE, T.INTEG-APPLI-CODE.LOAD, T.INTEG-APPLI-DATA, T.INTEG-APPLI-DATA.LOAD, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA

Table 2 Security Objectives and Threats - Coverage

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

Table 3 OSPs and Security Objectives - Coverage

Security Objectives	Organisational Security Policies
O.SID	
O.FIREWALL	
O.GLOBAL_ARRAYS_CONFID	
O.GLOBAL_ARRAYS_INTEG	
O.ARRAY_VIEWS_CONFID	
O.ARRAY_VIEWS_INTEG	
O.NATIVE	
O.OPERATE	
O.REALLOCATION	
O.RESOURCES	
O.ALARM	
O.CIPHER	
O.RNG	
O.KEY-MNGT	
O.PIN-MNGT	
O.TRANSACTION	
O.OBJ-DELETION	
O.DELETION	
O.LOAD	OSP.VERIFICATION
O.INSTALL	
OE.CAP_FILE	
OE.CARD-MANAGEMENT	
OE.SCP.IC	
OE.SCP.RECOVERY	
OE.SCP.SUPPORT	
OE.VERIFICATION	OSP.VERIFICATION
OE.CODE-EVIDENCE	OSP.VERIFICATION

Table 4 Security Objectives and OSPs – Coverage

Assumptions	Security Objectives for the Operational Environment	Rationale
A.CAP_FILE	OE.CAP_FILE	Section 6.3.3
A.DELETION	OE.CARD-MANAGEMENT	Section 6.3.3
A.VERIFICATION	OE.VERIFICATION, OE.CODE-EVIDENCE	Section 6.3.3

Table 5

Assumptions and Security Objectives for the Operational Environment - Coverage

Security Objectives for the Operational Environment	Assumptions
OE. CAP_FILE	A.CAP_FILE
OE.CARD-MANAGEMENT	A.DELETION
OE.SCP.IC	
OE.SCP.RECOVERY	
OE.SCP.SUPPORT	
OE.VERIFICATION	A.VERIFICATION
OE.CODE-EVIDENCE	A.VERIFICATION

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

7 SECURITY REQUIREMENTS

7.1 SECURITY FUNCTIONAL REQUIREMENTS

This section states the security functional requirements for the Java Card System - Open configuration. For readability and for compatibility with previous versions, requirements are arranged into groups. All the groups defined in the table below apply to this Protection Profile.

Group	Description
Core (*CoreG*)	The CoreG contains the requirements concerning the runtime environment of the Java Card System that may also implement logical channels. This includes the firewall policy and the requirements related to the Java Card API. Logical channels are a Java Card specification version 2.2-3.1 feature, and optional for version 3.2.
Installation (*InstG*)	The InstG contains the security requirements concerning the installation of post-issuance applications. It does not address card management issues in the broad sense, but only those security aspects of the installation procedure that are related to applet execution.
Applet deletion (*ADELG*)	The ADELG contains the security requirements for erasing installed applets from the card, a feature introduced in Java Card specification version 2.2.
Object deletion (*ODELG*)	The ODELG contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card specification version 2.2 feature.
Secure carrier (*CarG*)	The CarG group contains minimal requirements for secure downloading of applications on the card. This group contains the security requirements for preventing, the installation of a CAP file that has not been bytecode verified, or that has been modified after bytecode verification.

The SFRs refer to all potentially applicable subjects, objects, information, operations and security attributes.

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

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

Subject	Description
S.ADEL	The applet deletion manager which also acts on behalf of the card issuer. It may be an applet ([JCRE3], §11), but its role asks anyway for a specific treatment from the security viewpoint.
S.APPLET	Any applet instance.
S.BCV	The bytecode verifier (BCV), which acts on behalf of the verification authority who is in charge of the bytecode verification of the CAP files..
S.CAD	The CAD represents off-card entity that communicates with the S.INSTALLER. If the TOE provides JCRMI functionality, CAD can request RMI services by issuing commands to the card.
S.INSTALLER	The installer is the on-card entity which acts on behalf of the card issuer. This subject is involved in the loading of CAP files and installation of applets.
S.JCRE	The runtime environment under which Java programs in a smart card are executed.
S.JCVM	The bytecode interpreter that enforces the firewall at runtime.
S.LOCAL	Operand stack of a JCVM frame, or local variable of a JCVM frame containing an object or an array of references.
S.MEMBER	Any object's field, static field or array position.
S.CAP_FILE	A CAP file may contain multiple Java language packages. A package is a namespace within the Java programming language that may contain classes and interfaces. A CAP file may contain packages that define either user library, or one or several applets. A CAP file compliant with Java Card Specifications version 3.1 may contain multiple Java language packages. An EXTENDED CAP file as specified in Java Card Specifications version 3.1 may contain only applet packages, only library packages or a combination of library packages. A COMPACT CAP file as specified in Java Card Specifications version 3.1 or CAP files compliant to previous versions of Java Card Specification, MUST contain only a single package representing a library or one or more applets.

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

Object	Description
O.APPLET	Any installed applet, its code and data.
O.CODE_CAP_FILE	The code of a CAP file, including all linking information. On the Java Card platform, a CAP file is the installation unit.
O.JAVAOBJECT	Java class instance or array. It should be noticed that KEYS, PIN, arrays and applet instances are specific objects in the Java programming language.

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

Information	Description
I.APDU	Any APDU sent to or from the card through the communication channel.
I.DATA	JCVM Reference Data: objectref addresses of APDU buffer, JCRE-owned instances of APDU class and byte array for install method.

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

Security attribute	Description/Value
Active Applets	The set of the active applets' AIDs. An active applet is an applet that is selected on at least one of the logical channels.
Applet Selection Status	"Selected" or "Deselected".
Applet's version number	The version number of an applet indicated in the export file.
CAP File AID	The AID of a CAP file.
Context	CAP file AID or "Java Card RE".
Currently Active Context	CAP file AID or "Java Card RE".
Dependent package AID	Allows the retrieval of the package AID and Applet's version number ([JCVM3], §4.5.2).
LC Selection Status	Multiselectable, Non-multiselectable or "None".
LifeTime	CLEAR_ON_DESELECT or PERSISTENT (*).
Owner	The Owner of an object is either the applet instance that created the object or the CAP file (library) where it has been defined (these latter objects can only be arrays that initialize static fields of the CAP file). The owner of a remote object is the applet instance that created the object.
Package AID	The AID of each package indicated in the export file.
Registered Applets	The set of AID of the applet instances registered on the card.
Resident CAP files	The set of AIDs of the CAP files already loaded on the card.
Resident packages	The set of AIDs of the packages already loaded on the card.
Selected Applet Context	CAP file AID or "None".

Sharing	Standard, SIO, Array View, Java Card RE entry point or global array.
Static References	Static fields of a CAP file may contain references to objects. The Static References attribute records those references.

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

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

Operation	Description
OP.ARRAY_ACCESS(O.JAVAOBJECT, field)	Read/Write an array component.
OP.ARRAY_LENGTH (O.JAVAOBJECT, field)	Get length of an array component.
OP.ARRAY_T_ALOAD(O.JAVAOBJECT, field)	Read from an array component
OP.ARRAY_T_ASTORE(O.JAVAOBJECT, field)	Write to an array component
OP.ARRAY_AASTORE(O.JAVAOBJECT, field)	Store into reference array component
OP.CREATE(Sharing, LifeTime) (*)	Creation of an object (new, makeTransient or createArrayView call).
OP.DELETE_APPLET(O.APPLET,...)	Delete an installed applet and its objects, either logically or physically.
OP.DELETE_CAP_FILE(O.CODE_CAP_FILE,...)	Delete a CAP file, either logically or physically.
OP.DELETE_CAP_FILE_APPLET(O.CODE_CAP_FILE,...)	Delete a CAP file and its installed applets, either logically or physically.
OP.INSTANCE_FIELD(O.JAVAOBJECT, field)	Read/Write a field of an instance of a class in the Java programming language.
OP.INVK_VIRTUAL(O.JAVAOBJECT, method, arg1,...)	Invoke a virtual method (either on a class instance or an array object).
OP.INVK_INTERFACE(O.JAVAOBJECT, method, arg1,...)	Invoke an interface method.
OP.JAVA(...)	Any access in the sense of [JCRE3], §6.2.8. It stands for one of the operations

	OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW, OP.TYPE_ACCESS. OP.ARRAY_LENGTH
OP.PUT(S1,S2,I)	Transfer a piece of information I from S1 to S2.
OP.THROW(O.JAVAOBJECT)	Throwing of an object (athrow, see [JCRE3], §6.2.8.7).
OP.TYPE_ACCESS(O.JAVAOBJECT, class)	Invoke checkcast or instance of on an object in order to access to classes (standard or shareable interfaces objects).

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

7.1.1 CoreG Security Functional Requirements

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

7.1.1.1 Firewall Policy

FDP_ACC.2/FIREWALL Complete access control

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

Refinement:

The operations involved in the policy are:

- OP.CREATE,
- OP.INVK_INTERFACE,
- OP.INVK_VIRTUAL,
- OP.JAVA,
- OP.THROW,
- OP.TYPE_ACCESS.
- OP.ARRAY_LENGTH
- OP.ARRAY_T_ALOAD
- OP.ARRAY_T_ASTORE
- OP.ARRAY_AASTORE

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

Application Note:

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

FDP_ACF.1/FIREWALL Security attribute based access control

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

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

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

- **R.JAVA.1 ([JCRE3], §6.2.8): S.CAP_FILE may freely perform, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW or OP.TYPE_ACCESS upon any O.JAVAOBJECT whose Sharing attribute has value "JCRE entry point" or "global array".**
- **R.JAVA.2 ([JCRE3], §6.2.8): S.CAP_FILE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE or OP.THROW upon any O.JAVAOBJECT whose Sharing attribute has value "Standard" and whose Lifetime attribute has value "PERSISTENT" only if O.JAVAOBJECT's Context attribute has the same value as the active context.**
- **R.JAVA.3 ([JCRE3], §6.2.8.10): S. CAP_FILE may perform OP.TYPE_ACCESS upon an O.JAVAOBJECT with Context attribute different from the currently active context, 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 ([JCRE3], §6.2.8.6): S.CAP_FILE may perform OP.INVK_INTERFACE upon an O.JAVAOBJECT with Context attribute different from the currently active context, whose Sharing attribute has the value "SIO", and whose Context attribute has the value "CAP File AID", only if the invoked interface method extends the Shareable interface and one of the following conditions applies:**

a) The value of the attribute Selection Status of the CAP file whose AID is "CAP File AID" is "Multiselectable",

b) The value of the attribute Selection Status of the CAP file whose AID is "CAP File AID" is "Non-multiselectable", and either "CAP File AID" is the value of the currently selected applet or otherwise "CAP File AID" does not occur in the attribute Active Applets.

- **R.JAVA.5: S.CAP_FILE may perform OP.CREATE upon O.JAVAOBJECT only if the value of the Sharing parameter is "Standard" or "SIO".**
- **R.JAVA.6 ([JCRE3], §6.2.8): S.CAP_FILE may freely perform OP.ARRAY_ACCESS or OP.ARRAY_LENGTH upon any O.JAVAOBJECT whose Sharing attribute has value "global array".**

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

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

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

- **1) Any subject with OP.JAVA upon an O.JAVAOBJECT whose LifeTime attribute has value "CLEAR_ON_DESELECT" if O.JAVAOBJECT's Context attribute is not the same as the Selected Applet Context.**
- **2) Any subject attempting to create an object by the means of OP.CREATE and a "CLEAR_ON_DESELECT" LifeTime parameter if the active context is not the same as the Selected Applet Context**.
- **3) S.CAP_FILE performing OP.ARRAY_AASTORE of the reference of an O.JAVAOBJECT whose sharing attribute has value "global array" or "Temporary".**
- **4) S.CAP_FILE performing OP.PUTFIELD or OP.PUTSTATIC of the reference of an O.JAVAOBJECT whose sharing attribute has value "global array" or "Temporary"**
- **5) R.JAVA.7 ([JCRE3], §6.2.8.2): S.CAP_FILE performing OP.ARRAY_T_ASTORE into an array view without ATTR_WRITABLE_VIEW access attribute.**
- **6) R.JAVA.8 ([JCRE3], §6.2.8.2):S.CAP_FILE performing OP.ARRAY_T_ALOAD into an array view without ATTR_READABLE_VIEW access attribute.**

*Application Note:*FDP_ACF.1.4/FIREWALL:

- The deletion of applets 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.

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

The Sharing attribute defines five categories of objects:

- Standard ones, whose both fields and methods are under the firewall policy,
- Shareable interface Objects (SIO), which provide a secure mechanism for inter-applet communication,
- JCRE entry points (Temporary or Permanent), who have freely accessible methods but protected fields,
- Global arrays, having both unprotected fields (including components; refer to JavaCardClass discussion above) and methods.
- Array Views, having fields/elements access controlled by access control attributes, ATTR_READABLE_VIEW and ATTR_WRITABLE_VIEW 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 ([JCRE3], §6.1.3). An object is owned by an applet instance, by the JCRE or by the library where it has been defined (these latter objects can only be arrays that initialize static fields of CAP files).

([JCRE3], Glossary) Selected Applet Context. The Java Card RE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command with this applet's AID, the Java Card RE makes this applet the Selected Applet Context. The Java Card RE sends all APDU commands to the Selected Applet Context.

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

([JCRE3], §6.1.2.1) At any point in time, there is only one active context within the Java Card VM (this is called the Currently Active Context).

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

It should be noticed that the Java Card platform, version 2.2.x and version 3.x.x Classic Edition, introduces the possibility for an applet instance to be selected on multiple logical channels at the same time, or accepting other applets belonging to the same CAP file being selected simultaneously. These applets are referred to as multiselectable applets. Applets that belong to a same CAP file are either all multiselectable or not ([JCVM3], §2.2.5). Therefore, the selection mode can be regarded as an attribute of CAP files. No selection mode is defined for a library CAP file.

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 ([JCRE3], §4).

FDP_IFC.1/JCVM Subset information flow control

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

Application Note:

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

FDP_IFF.1/JCVM Simple security attributes

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

Subjects	Security attributes
S.JCVM	Currently Active Context

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

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

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

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

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

Application Note:

The storage of temporary Java Card RE-owned objects references is runtime-enforced ([JCRE3], §6.2.8.1-3).

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

FDP_RIP.1/OBJECTS Subset residual information protection

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

Application Note:

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

FMT_MSA.1/JCRE Management of security attributes

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

Application Note:

The modification of the Selected Applet Context should be performed in accordance with the rules given in [JCRE3], §4 and [JCVM3], §3.4.

FMT_MSA.1/JCVM Management of security attributes

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

Application Note:

The modification of the Currently Active Context should be performed in accordance with the rules given in [JCRE3], §4 and [JCVM3], §3.4.

FMT_MSA.2/FIREWALL_JCVM Secure security attributes

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

Application Note:

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

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

FMT_MSA.3/FIREWALL Static attribute initialisation

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

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

Application Note:

FMT_MSA.3.1/FIREWALL

- Objects' security attributes of the access control policy are created and initialized at the creation of the object or the subject. Afterwards, these attributes are no longer mutable (FMT_MSA.1/JCRE). At the creation of an object (OP.CREATE), the newly created object, assuming that the FIREWALL access control SFP permits the operation, gets its Lifetime and Sharing attributes from the parameters of the operation; on the contrary, its Context attribute has a default value, which is its creator's Context attribute and AID respectively ([JCRE3], §6.1.3). There is one default value for the Selected Applet Context that is the default applet identifier's Context, and one default value for the Currently Active Context that is "Java Card RE".
- The knowledge of which reference corresponds to a temporary entry point object or a global array and which does not is solely available to the Java Card RE (and the Java Card virtual machine).

FMT_MSA.3.2/FIREWALL

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

FMT_MSA.3/JCVM Static attribute initialisation

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

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

FMT_SMF.1 Specification of Management Functions

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

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

FMT_SMR.1 Security roles

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

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

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

7.1.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

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

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

FCS_CKM.1 Cryptographic key generation

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

Application Note:

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

Refer to Appendix 3 to define the allowed/available key generation algorithms as per Java Card API specifications [JCAPI3]. The ST Author should choose the algorithm

implemented to perform key generation. For each algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm.

FCS_CKM.6 Timing and event of cryptographic key destruction

FCS_CKM.6.1 The TSF shall destroy **[assignment: list of cryptographic keys (including keying material)]** when **[selection: no longer needed, [assignment: other circumstances for key or keying material destruction]]**.

FCS_CKM.6.2 The TSF shall destroy cryptographic keys and keying material specified by FCS_CKM.6.1 in accordance with a specified cryptographic key destruction method **[assignment: cryptographic key destruction method]** that meets the following: **[assignment: list of standards]**.

Application Note:

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

FCS_COP.1 Cryptographic operation

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

Application Note:

Refer to Appendix 3 to define the allowed/available algorithms as per Java Card API specifications [JCAPI3] The ST Author should choose the algorithm implemented to perform crypto operations. For each algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm.

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

Random Numbers

FCS_RNG.1 Random number generation

FCS_RNG.1.1 The TSF shall provide a **[selection: physical, non-physical true, deterministic, hybrid physical, hybrid deterministic]** random number generator that implements: **[assignment: list of security capabilities].**

FCS_RNG.1.2 The TSF shall provide **[selection: bits, octets of bits, numbers [assignment: format of the numbers]]** that meet **[assignment: a defined quality metric]**.

Application Note:

The ST writer shall perform the open operations. The operation performed in the element FCS_RNG.1.1 selects RNG types based on physical random number generators as typical provided by Security IC.

FDP_RIP.1/ABORT Subset residual information protection

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

Application Note:

The events that provoke the de-allocation of a transient object are described in [JCRE3], §5.1.

FDP_RIP.1/APDU Subset residual information protection

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

Application Note:

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

FDP_RIP.1/GlobalArray Subset residual information protection

FDP_RIP.1.1/GlobalArray [*Refined*]
The TSF shall ensure that any previous information content of a resource is made unavailable upon **deallocation of the resource from** *the applet as a result of returning from the process method to* the following objects: **a user Global Array**.

Application Note:

An array resource is allocated when a call to the API method JCSystem.makeGlobalArray is performed. The Global Array is created as a transient JCRE Entry Point Object ensuring that reference to it cannot be retained by any application. On return from the method which called JCSystem.makeGlobalArray, the array is no longer available to any applet and is deleted and the memory in use by the array is cleared and reclaimed in the next object deletion cycle.

FDP_RIP.1/bArray Subset residual information protection

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

Application Note:

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

FDP_RIP.1/KEYS Subset residual information protection

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

Application Note:

- The javacard.security & javacardx.crypto packages do provide secure interfaces to the cryptographic buffer in a transparent way. See javacard.security.KeyBuilder and Key interface of [JCAPI3].

FDP_RIP.1/TRANSIENT Subset residual information protection

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

Application Note:

- The events that provoke the de-allocation of any transient object are described in [JCRE3], §5.1.

- The clearing of CLEAR_ON_DESELECT objects is not necessarily performed when the owner of the objects is deselected. In the presence of multiselectable applet instances, CLEAR_ON_DESELECT memory segments may be attached to applets that are active in different logical channels. Multiselectable applet instances within a same CAP file must share the transient memory segment if they are concurrently active ([JCRE3], §4.3[8].

FDP_ROL.1/FIREWALL Basic rollback

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

FDP_ROL.1.2/FIREWALL The TSF shall permit operations to be rolled back within the **scope of a select(), deselect(), process(), install() or uninstall() call, notwithstanding the restrictions given in [JCRE3], §7.7, within the bounds of the Commit Capacity ([JCRE3], §7.8), and those described in [JCAPI3]**.

Application Note:

Transactions are a service offered by the APIs to applets. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [JCAPI3] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).

7.1.1.3 Card Security Management

FAU_ARP.1 Security alarms

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

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

upon detection of a potential security violation.

Refinement:

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

- CAP file inconsistency,
- typing error in the operands of a bytecode,

[8] [8]Section §4.3 for [JCRE3] versions 3.0.4, 3.0.5 and 3.1. Section §4.2 for [JCRE3] version 3.0.1

- applet life cycle inconsistency,
- card tearing (unexpected removal of the Card out of the CAD) and power failure,
- abort of a transaction in an unexpected context, (see abortTransaction(), [JCAPI3] and ([JCRE3], §7.6.2)
- violation of the Firewall or JCVM SFPs,
- unavailability of resources,
- array overflow,
- [assignment: list of other runtime errors].

Application Note:

- The developer shall provide the exhaustive list of actual potential security violations the TOE reacts to. For instance, other runtime errors related to applet's failure like uncaught exceptions.
- The bytecode verification defines a large set of rules used to detect a "potential security violation". The actual monitoring of these "events" within the TOE only makes sense when the bytecode verification is performed on-card.
- Depending on the context of use and the required security level, there are cases where the card manager and the TOE must work in cooperation to detect and appropriately react in case of potential security violation. This behavior must be described in this component. It shall detail the nature of the feedback information provided to the card manager (like the identity of the offending application) and the conditions under which the feedback will occur (any occurrence of the java.lang.SecurityException exception).
- The "locking of the card session" may not appear in the policy of the card manager. Such measure should only be taken in case of severe violation detection; the same holds for the re-initialization of the Java Card System. Moreover, the locking should occur when "clean" re-initialization seems to be impossible.
- The locking may be implemented at the level of the Java Card System as a denial of service (through some systematic "fatal error" message or return value) that lasts up to the next "RESET" event, without affecting other components of the card (such as the card manager). Finally, because the installation of applets is a sensitive process, security alerts in this case should also be carefully considered herein.

FDP_SDI.2/DATA Stored data integrity monitoring and action

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

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

Application Note:

- Although no such requirement is mandatory in the Java Card specification, at least an exception shall be raised upon integrity errors detection on cryptographic keys, PIN values and their associated security attributes. Even if all the objects cannot be

monitored, cryptographic keys and PIN objects shall be considered with particular attention by ST authors as they play a key role in the overall security.

- It is also recommended to monitor integrity errors in the code of the native applications and Java Card applets.

For integrity sensitive application, their data shall be monitored (D.APP_I_DATA): applications may need to protect information against unexpected modifications, and explicitly control whether a piece of information has been changed between two accesses. For example, maintaining the integrity of an electronic purse's balance is extremely important because this value represents real money. Its modification must be controlled, for illegal ones would denote an important failure of the payment system.

- A dedicated library could be implemented and made available to developers to achieve better security for specific objects, following the same pattern that already exists in cryptographic APIs, for instance.

FPR_UNO.1 Unobservability

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

Application Note:

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 values when they exist on the card, as well as the cryptographic operations and comparisons performed on them.

FPT_FLS.1 Failure with preservation of secure state

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

Application Note:

The Java Card RE Context is the Current context when the Java Card VM begins running after a card reset ([JCRE3], §6.2.3) or after a proximity card (PICC) activation sequence ([JCRE3]). Behavior of the TOE on power loss and reset is described in [JCRE3], §3.6 and §7.1. Behavior of the TOE on RF signal loss is described in [JCRE3], §3.6.1.

FPT_TDC.1 Inter-TSF basic TSF data consistency

FPT_TDC.1.1 The TSF shall provide the capability to consistently interpret **the CAP files, the bytecode and its data arguments** when shared between the TSF and another trusted IT product.

FPT_TDC.1.2 The TSF shall use

- **the rules defined in [JCVM3] specification,**
- **the API tokens defined in the export files of reference implementation,**
- **[assignment: list of interpretation rules to be applied by the TSF]**

when interpreting the TSF data from another trusted IT product.

Application Note:

Concerning the interpretation of data between the TOE and the underlying Java Card platform, it is assumed that the TOE is developed consistently with the SCP functions, including memory management, I/O functions and cryptographic functions.

7.1.1.4 AID MANAGEMENT

FIA_ATD.1/AID User attribute definition

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

- **CAP File AID,**
- **Package AID,**
- **Applet's version number,**
- **Registered applet AID,**
- **Applet Selection Status**.

Refinement:

"Individual users" stand for applets.

FIA_UID.2/AID User identification before any action

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

Application Note:

- By users here it must be understood the ones associated to the CAP files (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 CAP file that is the subject's owner. Means of identification are provided during the loading procedure of the CAP file and the registration of applet instances.

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

FIA_USB.1/AID User-subject binding

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

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

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

Application Note:

The user is the applet and the subject is the S.CAP_FILE. The subject security attribute "Context" shall hold the user security attribute "CAP file AID".

FMT_MTD.1/JCRE Management of TSF data

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

Application Note:

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

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

7.1.2 INSTG SECURITY FUNCTIONAL REQUIREMENTS

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

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

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

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.

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.

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

CAP file loading is allowed only if, for each dependent package, its AID attribute is equal to a resident package AID attribute, the major version attribute associated to the dependent package file is equal to the major version attribute of the resident package and the minor version attribute is equal to or less than the minor version attribute associated to the resident package ([JCVM3], §4.5.2).

Application Note:

FDP_ITC.2.1/Installer:

- The most common importation of user data is CAP file 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 CAP file and the package or packages contained within the CAP file, maximal operand stack size and number of local variables for each method, and export and import components (accessibility).

FDP_ITC.2.3/Installer:

- The format of the CAP file is precisely defined in [JCVM3] specifications; it contains the user data (like applet's code and data) and the security attributes altogether. Therefore there is no association to be carried out elsewhere.

FDP_ITC.2.4/Installer:

- Each CAP file and all the packages contained within a CAP file contain a Version attribute, which is a pair of major and minor version numbers ([JCVM3], §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 version numbers and AIDs of imported packages indicated in the export file are recorded in the CAP files ([JCVM3], §4.5.2): the dependent packages' Version and AID attributes allow the retrieval of these identifications. Implementation-dependent checks may occur on a case-by-case basis to check that packages are binary compatible. Packages have "package Version Numbers" ([JCVM3]) that indicate binary compatibility or incompatibility between successive implementations of a package, which directly concern this requirement.

FDP_ITC.2.5/Installer:

- A package may depend on (import or use data from) other packages already installed. This dependency is explicitly stated in the loaded package in the form of a list of package AIDs.
- The intent of this rule is to ensure the binary compatibility of the package with those already on the card ([JCVM3], §4.4).
- The installation (the invocation of an applet's install method by the installer) is implementation dependent ([JCRE3], §11.2).
- 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, [JCRE3], §11).

FMT_SMR.1/Installer Security roles

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

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

FPT_FLS.1/Installer 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 CAP file/applet as described in [JCRE3] §11.1.5**.

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/Installer 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 to a secure state is provided.

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.

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 under the control of the TSF.

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

Application Note:

FPT_RCV.3.1/Installer:

- 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 [CC2], p298: In this maintenance mode normal operation might be impossible or severely restricted, as otherwise insecure situations might occur. Typically, only authorised users should be allowed access to this mode but the real details of who can access this mode is a function of FMT: Security management. If FMT: Security management 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 SFRs.

FPT_RCV.3.2/Installer:

- Should the installer fail during loading/installation of a CAP file/applet, it has to revert to a "consistent and secure state". The Java Card RE has some clean up duties as well; see [JCRE3], §11.1.5 for possible scenarios. Precise behavior is left to implementers. This

component shall include among the listed failures the deletion of a CAP file/applet. See ([JCRE3], 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 [PP0084b] or [PP117]) and, from the TOE's side, by events "that clear transient objects" and transactional features. See FPT_FLS.1.1, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ABORT and FDP_ROL.1/FIREWALL.

FPT_RCV.3.3/Installer:

- The quantification is implementation dependent, but some facts can be recalled here. First, the SCP ensures the atomicity of updates for fields and objects, 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 TSF scope should be limited to the same restrictions of the transaction mechanism.

7.1.3 ADELG Security Functional Requirements

This group consists of the SFRs related to the deletion of applets and/or CAP files, enforcing the applet deletion manager (ADEL) policy on security aspects outside the runtime. Deletion is a critical operation and therefore requires specific treatment. This policy is better thought as a frame to be filled by ST implementers.

FDP_ACC.2/ADEL Complete access control

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

Refinement:

The operations involved in the policy are:

- OP.DELETE_APPLET,
- OP.DELETE_CAP_FILE,
- OP.DELETE_CAP_FILE_APPLET.

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

FDP_ACF.1/ADEL 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:

Subject/Object	Attributes
S.JCVM	Active Applets
S.JCRE	Selected Applet Context, Registered Applets, Resident CAP files
O.CODE_CAP_FILE	CAP file AID, AIDs of packages within a CAP file, Dependent package AID, Static References
O.APPLET	Applet Selection Status
O.JAVAOBJECT	Owner, Remote

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

In the context of this policy, an object O is reachable if and only one of the following conditions hold:

- **(1) the owner of O is a registered applet instance A (O is reachable from A),**
- **(2) a static field of a resident package P contains a reference to O (O is reachable from P),**
- **(3) there exists a valid remote reference to O (O is remote reachable),**
- **(4) there exists an object O' that is reachable according to either (1) or (2) or (3) above and O' contains a reference to O (the reachability status of O is that of O').**

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

- **R.JAVA.14 ([JCRE3], §11.3.4.2[9], Applet Instance Deletion): S.ADEL may perform OP.DELETE_APPLET upon an O.APPLET only if,**

(1) S.ADEL is currently selected,

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

(3) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance distinct from O.APPLET, or O.JAVAOBJECT is reachable from a package P, or ([JCRE3], §8.5) O.JAVAOBJECT is remote reachable.

- **R.JAVA.15 ([JCRE3], §11.3.4.2.1[10], Multiple Applet Instance Deletion): S.ADEL may perform OP.DELETE_APPLET upon several O.APPLET only if,**

[9] Section §11.3.4.2 for [JCRE3] versions 3.0.4, 3.0.5, 3.1 and 3.2. Section §11.3.4.1 for [JCRE] version 3.0.1
[10] Section §11.3.4.2.1 for [JCRE3] versions 3.0.4, 3.0.5, 3.1 and 3.2. Section §11.3.4.1 for [JCRE] version 3.0.1

(1) S.ADEL is currently selected,

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

(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 CAP file P, or ([JCRE3], §8.5) O.JAVAOBJECT is remote reachable.

- **R.JAVA.16 ([JCRE3], §11.3.4.3[11], Applet/Library CAP file Deletion): S.ADEL may perform OP.DELETE_CAP_FILE upon an O.CODE_CAP_FILE only if,**

(1) S.ADEL is currently selected,

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

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

- **R.JAVA.17 ([JCRE3], §11.3.4.4[12], Applet CAP file and Contained Instances Deletion): S.ADEL may perform OP.DELETE_CAP_FILE_APPLET upon an O.CODE_CAP_FILE only if,**

(1) S.ADEL is currently selected,

(2) no reachable O.JAVAOBJECT, from a CAP file distinct from O.CODE_CAP_FILE, which is an instance of a class that belongs to O.CODE_CAP_FILE exists on the card,

(3) there is no CAP file loaded on the card that depends on O.CODE_CAP_FILE, and

(4) for every O.APPLET of those being deleted it holds that: (i) there is no instance in the context of O.APPLET that is active in any logical channel and (ii) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance not being deleted, or O.JAVAOBJECT is reachable from a CAP file not being deleted, or ([JCRE3], §8.5) O.JAVAOBJECT is remote reachable.

[11] Section §11.3.4.3 for [JCRE3] versions 3.0.4, 3.0.5, 3.1 and 3.2. Section §11.3.4.2 for [JCRE] version 3.0.1
[12] Section §11.3.4.4 for [JCRE3] versions 3.0.4, 3.0.5, 3.1 and 3.2. Section §11.3.4.3 for [JCRE] version 3.0.1

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

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

any subject but S.ADEL to O.CODE_PKG or O.APPLET for the purpose of deleting them from the card.

Application Note:

FDP_ACF.1.2/ADEL:

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

FDP_RIP.1/ADEL 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 **deallocation of the resource from** the following objects: **applet instances and/or CAP files 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/CAP file deletion are described in [JCRE3], §11.3.4.1, §11.3.4.2 and §11.3.4.3.

FMT_MSA.1/ADEL 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 security attributes **Registered Applets and Resident CAP files** to **the Java Card RE**.

FMT_MSA.3/ADEL Static attribute initialisation

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): none,** to specify alternative initial values to override the default values when an object or information is created.

FMT_SMF.1/ADEL Specification of Management Functions

FMT_SMF.1.1/ADEL The TSF shall be capable of performing the following management functions: **modify the list of registered applets' AIDs and the Resident CAP files**.

FMT_SMR.1/ADEL Security roles

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

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

FPT_FLS.1/ADEL 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 CAP file/applet as described in [JCRE3], §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).
- The CAP file/applet instance deletion must be atomic. The "secure state" referred to in the requirement must comply with Java Card specification ([JCRE3], §11.3.4.)

7.1.4 ODELG SECURITY FUNCTIONAL REQUIREMENTS

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

FDP_RIP.1/ODEL Subset residual information protection

FDP_RIP.1.1/ODEL The TSF shall ensure that any previous information content of a resource is made unavailable upon the **deallocation of the resource from** the following objects: **the objects owned by the context of an applet instance which triggered**

the execution of the method
`javacard.framework.JCSystem.requestObjectDeletion()`.

Application Note:

- Freed data resources resulting from the invocation of the method javacard.framework.JCSystem.requestObjectDeletion() may be reused. Requirements on de-allocation after the invocation of the method are described in [JCAPI3].
- There is no conflict with FDP_ROL.1 here because of the bounds on the rollback mechanism: the execution of requestObjectDeletion() is not in the scope of the rollback because it must be performed in between APDU command processing, and therefore no transaction can be in progress.

FPT_FLS.1/ODEL Failure with preservation of secure state

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

Application Note:

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

7.1.5 CarG Security Functional Requirements

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

FCO_NRO.2/CM Enforced proof of origin

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

FCO_NRO.2.2/CM The TSF shall be able to relate the **identity** of the originator of the information, and the **application CAP file,** of the information to which the evidence applies.

FCO_NRO.2.3/CM The TSF shall provide a capability to verify the evidence of origin of information to **recipient** given **[assignment: limitations on the evidence of origin]**.

Application Note:

FCO_NRO.2.1/CM:

- Upon reception of a new application CAP file for installation, the card manager shall first check that it actually comes from the verification authority and represented by the subject S.BCV. The verification authority is indeed the entity responsible for bytecode verification.

FCO_NRO.2.3/CM:

- 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 CAP file using an electronic signature mechanism, and no evidence is kept on the card for future verifications.

FDP_IFC.2/CM Complete information flow control

FDP_IFC.2.1/CM The TSF shall enforce the **CAP FILE LOADING information flow control SFP** on **S.INSTALLER, S.BCV, S.CAD and I.APDU** and all operations that cause that information to flow to and from subjects covered by the SFP.

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

Application Note:

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

FDP_IFF.1/CM Simple security attributes

FDP_IFF.1.1/CM The TSF shall enforce the **CAP FILE LOADING 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.1.2/CM The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: **[assignment:**

the rules describing the communication protocol used by the CAD and the card for transmitting a new CAP file].

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 explicitly authorise an information flow based on the following rules: **[assignment: rules, based on security attributes, that explicitly authorise information flows]**.

FDP_IFF.1.5/CM The TSF shall explicitly deny an information flow based on the following rules:

- **The TOE fails to verify the integrity and authenticity evidences of the application CAP file.**
- **[assignment: rules, based on security attributes, that explicitly deny information flows]**.

Application Note:

FDP_IFF.1.1/CM:

- The security attributes used to enforce the CAP FILE 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 CAP file has been split into in order to be sent to the card; (3) the ordinal of each piece in the decomposition of the CAP file, etc. See for example Appendix D of [GP].

FDP_IFF.1.2/CM:

- 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.INSTALLER shall accept a message only if it comes from the subject S.CAD; (2) the subject S.INSTALLER shall accept an application CAP file only if it has received without modification and in the right order all the APDUs sent by the subject S.CAD.

FDP_IFF.1.5/CM:

- The verification of the integrity and authenticity evidences can be performed either during loading or during the first installation of an application of the CAP file.

FDP_UIT.1/CM Data exchange integrity

FDP_UIT.1.1/CM The TSF shall enforce the **CAP FILE LOADING information flow control SFP** to **[selection: transmit, receive]** user data in a manner protected from **[selection: modification, deletion, insertion, replay]** errors.

FDP_UIT.1.2/CM [*Refined*] The TSF shall be able to determine on receipt of user data, whether **modification, deletion, insertion, *replay of some of the pieces of the application sent by the CAD*** has occurred.

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 CAP file to be installed on the card to be different from the one sent by the CAD.

FIA_UID.1/CM 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 CAP file 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.

FMT_MSA.1/CM Management of security attributes

FMT_MSA.1.1/CM The TSF shall enforce the **CAP FILE 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 authorised identified roles]**.

FMT_MSA.3/CM Static attribute initialisation

FMT_MSA.3.1/CM The TSF shall enforce the **CAP FILE 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 authorised identified roles]** to specify alternative initial values to override the default values when an object or information is created.

FMT_SMF.1/CM Specification of Management Functions

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

FMT_SMR.1/CM Security roles

FMT_SMR.1.1/CM The TSF shall maintain the roles **[assignment: the authorised identified roles]**.

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

FTP_ITC.1/CM Inter-TSF trusted channel

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

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

FTP_ITC.1.3/CM The TSF shall initiate communication via the trusted channel for **loading/installing a new application CAP file on the card**.

Application Note:

There is no dynamic CAP file loading on the Java Card platform. New CAP files can be installed on the card only on demand of the card issuer.

7.2 SECURITY ASSURANCE REQUIREMENTS

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

7.3 SECURITY REQUIREMENTS RATIONALE

7.3.1 OBJECTIVES

7.3.1.1 SECURITY OBJECTIVES FOR THE TOE

7.3.1.1.1 IDENTIFICATION

O.SID Subjects' identity is AID-based (applets, packages and CAP files), and is met by the following SFRs: FDP_ITC.2/Installer, FIA_ATD.1/AID, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_MSA.1/ ADEL, FMT_MSA.1/CM, FMT_MSA.3/ADEL, FMT_MSA.3/FIREWALL, FMT_MSA.3/JCVM, FMT_MSA.3/CM, FMT_SMF.1/CM, FMT_SMF.1/ADEL, FMT_SMF.1/ADEL, FMT_MTD.1/JCRE and FMT_MTD.3/JCRE.

Installation procedures ensure protection against forgery (the AID of an applet is under the control of the TSFs) or re-use of identities (FIA_UID.2/AID, FIA_USB.1/AID).

7.3.1.1.2 EXECUTION

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

O.GLOBAL_ARRAYS_CONFID Only arrays can be designated as global, and the only global arrays required in the Java Card API are the APDU buffer, the global byte array input parameter (bArray) to an applet's install method and the global arrays created by the JCSystem.makeGlobalArray(…) method. The clearing requirement of these arrays is met by (FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray and FDP_RIP.1/bArray respectively). The JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM) prevents an application from keeping a pointer to a shared buffer, which could be used to read its contents when the buffer is being used by another application.

If the TOE provides JCRMI functionality, protection of the array parameters of remotely invoked methods, which are global as well, is covered by the general initialization of method parameters (FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/ADEL and FDP_RIP.1/TRANSIENT).

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

O.ARRAY_VIEWS_CONFID Array views have security attributes of temporary objects where the JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM) prevents

an application from storing a reference to the array view. Furthermore, array views may not have ATTR_READABLE_VIEW security attribute which ensures that no application can read the contents of the array view.

O.ARRAY_VIEWS_INTEG Array views have security attributes of temporary objects where the JCVM information flow control policy (FDP_IFF.1/JCVM, FDP_IFC.1/JCVM) prevents an application from storing a reference to the array view. Furthermore, array views may not have ATTR_WRITABLE_VIEW security attribute which ensures that no application can alter the contents of the array view.

O.NATIVE This security objective is covered by FDP_ACF.1/FIREWALL: the only means to execute native code is the invocation of a Java Card API method. This objective mainly relies on the environmental objective OE.**CAP_FILE**, which uphold the assumption A.CAP_FILE.

O.OPERATE The TOE is protected in various ways against applets' actions (FPT_TDC.1), the FIREWALL access control policy FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL, and is able to detect and block various failures or security violations during usual working (FPT_FLS.1/ADEL, FPT_FLS.1, FPT_FLS.1/ODEL, FPT_FLS.1/Installer, FAU_ARP.1). Its security-critical parts and procedures are also protected: safe recovery from failure is ensured (FPT_RCV.3/Installer), applets' installation may be cleanly aborted (FDP_ROL.1/FIREWALL), communication with external users and their internal subjects is well-controlled (FDP_ITC.2/Installer, FIA_ATD.1/AID, FIA_USB.1/AID) to prevent alteration of TSF data (also protected by components of the FPT class).

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

Application note: Startup of the TOE (TSF-testing) can be covered by FPT_TST.1. This SFR component is not mandatory in [JCRE3], but appears in most of security requirements documents for masked applications. Testing could also occur randomly. Self-tests may become mandatory in order to comply with FIPS certification [FIPS 140-2].

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

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

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

7.3.1.1.3 SERVICES

O.ALARM This security objective is met by FPT_FLS.1/Installer, FPT_FLS.1, FPT_FLS.1/ADEL, FPT_FLS.1/ODEL which guarantee that a secure state is preserved by the TSF when failures

occur, and FAU_ARP.1 which defines TSF reaction upon detection of a potential security violation.

O.CIPHER This security objective is directly covered by FCS_CKM.1, , FCS_CKM.6 and FCS_COP.1. The SFR FPR_UNO.1 contributes in covering this security objective and controls the observation of the cryptographic operations which may be used to disclose the keys.

O.RNG This security objective is directly covered by FCS_RNG.1 which ensures the cryptographic quality of random number generation.

O.KEY-MNGT This relies on the same security functional requirements as O.CIPHER, plus FDP_RIP.1 and FDP_SDI.2/DATA as well. Precisely it is met by the following components: FCS_CKM.1, FCS_CKM.6, FCS_COP.1, FPR_UNO.1, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/ADEL and FDP_RIP.1/TRANSIENT.

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

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

7.3.1.1.4 OBJECT DELETION

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

7.3.1.1.5 APPLET MANAGEMENT

O.DELETION This security objective specifies that applet and CAP file 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 CAP file 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/Installer). The security functional requirements of the class FMT (FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL) included in the group ADELG also contribute to meet this objective.

O.LOAD This security objective specifies that the loading of a CAP file into the card must be secure. Evidence of the origin of the CAP file is enforced (FCO_NRO.2/CM) and the integrity of the corresponding data is under the control of the CAP FILE LOADING information flow

policy (FDP_IFC.2/CM, FDP_IFF.1/CM) and FDP_UIT.1/CM. Appropriate identification (FIA_UID.1/CM) and transmission mechanisms are also enforced (FTP_ITC.1/CM).

O.INSTALL This security 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/Installer), and the TSFs are protected against possible failures of the installer (FPT_FLS.1/Installer, FPT_RCV.3/Installer).

7.3.2 RATIONALE TABLES OF SECURITY OBJECTIVES AND SFRS

Security Objectives	Security Functional Requirements	Rationale
O.SID	FIA_ATD.1/AID, FIA_UID.2/AID, FMT_MSA.1/JCRE, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_MSA.3/FIREWALL, FMT_MSA.1/CM, FMT_MSA.3/CM, FDP_ITC.2/Installer, FMT_SMF.1/CM, FMT_SMF.1/ADEL, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FIA_USB.1/AID, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM	Section 7.4.1.1.1
O.FIREWALL	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FMT_SMR.1/Installer, FMT_MSA.1/CM, FMT_MSA.3/CM, FMT_SMR.1/CM, FMT_MSA.3/FIREWALL, FMT_SMR.1, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL, FMT_MSA.1/JCRE, FDP_ITC.2/Installer, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FMT_SMF.1/ADEL, FMT_SMF.1/CM, FMT_SMF.1, FMT_MSA.2/FIREWALL_JCVM, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM	Section 7.4.1.1.2
O.GLOBAL_ARRAYS_CONFID	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FDP_RIP.1/bArray, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT	Section 7.4.1.1.2
O.GLOBAL_ARRAYS_INTEG	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM	Section 7.4.1.1.2
O.ARRAY_VIEWS_CONFID	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FDP_ACC.2/Firewall, FDP_ACF.1/Firewall	Section 7.4.1.1.2
O.ARRAY_VIEWS_INTEG	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FDP_ACC.2/Firewall, FDP_ACF.1/Firewall	Section 7.4.1.1.2
O.NATIVE	FDP_ACF.1/FIREWALL	Section 7.4.1.1.2

O.OPERATE	FAU_ARP.1, FDP_ROL.1/FIREWALL, FIA_ATD.1/AID, FPT_FLS.1/ADEL, FPT_FLS.1, FPT_FLS.1/ODEL, FPT_FLS.1/Installer, FDP_ITC.2/Installer, FPT_RCV.3/Installer, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FPT_TDC.1, FIA_USB.1/AID	Section 7.4.1.1.2
O.REALLOCATION	FDP_RIP.1/ABORT, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArrayFDP_RIP.1/bArray, FDP_RIP.1/KEYS, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ADEL, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS	Section 7.4.1.1.2
O.RESOURCES	FAU_ARP.1, FDP_ROL.1/FIREWALL, FMT_SMR.1/Installer, FMT_SMR.1, FMT_SMR.1/ADEL,FPT_FLS.1/Installer, FPT_FLS.1/ODEL, FPT_FLS.1, FPT_FLS.1/ADEL, FPT_RCV.3/Installer, FMT_SMR.1/CM, FMT_SMF.1/ADEL, FMT_SMF.1/CM, FMT_SMF.1, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE	Section 7.4.1.1.2
O.ALARM	FPT_FLS.1/Installer, FPT_FLS.1, FPT_FLS.1/ADEL, FPT_FLS.1/ODEL, FAU_ARP.1	Section 7.4.1.1.3
O.CIPHER	FCS_CKM.1, FCS_CKM.6, FCS_COP.1, FPR_UNO.1	Section 7.4.1.1.3
O.RNG	FCS_RNG.1	Section 7.4.1.1.3
O.KEY-MNGT	FCS_CKM.1, FCS_CKM.6, FCS_COP.1, FPR_UNO.1, FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_SDI.2/DATA, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT	Section 7.4.1.1.3
O.PIN-MNGT	FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FPR_UNO.1, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT, FDP_ROL.1/FIREWALL, FDP_SDI.2/DATA, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL	Section 7.4.1.1.3
O.TRANSACTION	FDP_ROL.1/FIREWALL, FDP_RIP.1/ABORT, FDP_RIP.1/ODEL, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray, FDP_RIP.1/bArray, FDP_RIP.1/KEYS, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT, FDP_RIP.1/OBJECTS	Section 7.4.1.1.3

O.OBJ-DELETION	FDP_RIP.1/ODEL, FPT_FLS.1/ODEL	Section 7.4.1.1.4
O.DELETION	FDP_ACC.2/ADEL, FDP_ACF.1/ADEL, FDP_RIP.1/ADEL, FPT_FLS.1/ADEL, FPT_RCV.3/Installer, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL	Section 7.4.1.1.5
O.LOAD	FCO_NRO.2/CM, FDP_IFC.2/CM, FDP_IFF.1/CM, FDP_UIT.1/CM, FIA_UID.1/CM, FTP_ITC.1/CM	Section 7.4.1.1.5
O.INSTALL	FDP_ITC.2/Installer, FPT_RCV.3/Installer, FPT_FLS.1/Installer	Section 7.4.1.1.5

Table 7 Security Objectives and SFRs - Coverage

Security Functional Requirements	Security Objectives
FDP_ACC.2/FIREWALL	O.FIREWALL, O.OPERATE, O.PIN-MNGT, O.ARRAY_VIEWS_CONFID, O.ARRAY_VIEWS_INTEG
FDP_ACF.1/FIREWALL	O.FIREWALL, O.NATIVE, O.OPERATE, O.PIN-MNGT, O.ARRAY_VIEWS_CONFID, O.ARRAY_VIEWS_INTEG
FDP_IFC.1/JCVM	O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.GLOBAL_ARRAYS_INTEG, O.ARRAY_VIEWS_CONFID, O.ARRAY_VIEWS_INTEG
FDP_IFF.1/JCVM	O.FIREWALL, O.GLOBAL_ARRAYS_CONFID, O.GLOBAL_ARRAYS_INTEG, O.ARRAY_VIEWS_CONFID, O.ARRAY_VIEWS_INTEG
FDP_RIP.1/OBJECTS	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FMT_MSA.1/JCRE	O.SID, O.FIREWALL
FMT_MSA.1/JCVM	O.SID, O.FIREWALL
FMT_MSA.2/FIREWALL_JCVM	O.FIREWALL
FMT_MSA.3/FIREWALL	O.SID, O.FIREWALL
FMT_MSA.3/JCVM	O.SID, O.FIREWALL
FMT_SMF.1	O.FIREWALL, O.RESOURCES

FMT_SMR.1	O.FIREWALL, O.RESOURCES
FCS_CKM.1	O.CIPHER, O.KEY-MNGT
FCS_CKM.6	O.CIPHER, O.KEY-MNGT
FCS_COP.1	O.CIPHER, O.KEY-MNGT
FCS_RNG.1	O_RNG
FDP_RIP.1/ABORT	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/APDU	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/bArray	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/GlobalArray	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/KEYS	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_RIP.1/TRANSIENT	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION
FDP_ROL.1/FIREWALL	O.OPERATE, O.RESOURCES, O.PIN-MNGT, O.TRANSACTION
FAU_ARP.1	O.OPERATE, O.RESOURCES, O.ALARM
FDP_SDI.2/DATA	O.KEY-MNGT, O.PIN-MNGT
FPR_UNO.1	O.CIPHER, O.KEY-MNGT, O.PIN-MNGT
FPT_FLS.1	O.OPERATE, O.RESOURCES, O.ALARM
FPT_TDC.1	O.OPERATE
FIA_ATD.1/AID	O.SID, O.OPERATE
FIA_UID.2/AID	O.SID
FIA_USB.1/AID	O.SID, O.OPERATE
FMT_MTD.1/JCRE	O.SID, O.FIREWALL, O.RESOURCES
FMT_MTD.3/JCRE	O.SID, O.FIREWALL, O.RESOURCES
FDP_ITC.2/Installer	O.SID, O.FIREWALL, O.OPERATE, O.INSTALL
FMT_SMR.1/Installer	O.FIREWALL, O.RESOURCES
FPT_FLS.1/Installer	O.OPERATE, O.RESOURCES, O.ALARM, O.INSTALL

FPT_RCV.3/Installer	O.OPERATE, O.RESOURCES, O.DELETION, O.INSTALL
FDP_ACC.2/ADEL	O.DELETION
FDP_ACF.1/ADEL	O.DELETION
FDP_RIP.1/ADEL	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION, O.DELETION
FMT_MSA.1/ADEL	O.SID, O.FIREWALL, O.DELETION
FMT_MSA.3/ADEL	O.SID, O.FIREWALL, O.DELETION
FMT_SMF.1/ADEL	O.SID, O.FIREWALL, O.RESOURCES
FMT_SMR.1/ADEL	O.FIREWALL, O.RESOURCES, O.DELETION
FPT_FLS.1/ADEL	O.OPERATE, O.RESOURCES, O.ALARM, O.DELETION
FDP_RIP.1/ODEL	O.GLOBAL_ARRAYS_CONFID, O.REALLOCATION, O.KEY-MNGT, O.PIN-MNGT, O.TRANSACTION, O.OBJ-DELETION
FPT_FLS.1/ODEL	O.OPERATE, O.RESOURCES, O.ALARM, O.OBJ-DELETION
FCO_NRO.2/CM	O.LOAD
FDP_IFC.2/CM	O.LOAD
FDP_IFF.1/CM	O.LOAD
FDP_UIT.1/CM	O.LOAD
FIA_UID.1/CM	O.LOAD
FMT_MSA.1/CM	O.SID, O.FIREWALL
FMT_MSA.3/CM	O.SID, O.FIREWALL
FMT_SMF.1/CM	O.SID, O.FIREWALL, O.RESOURCES
FMT_SMR.1/CM	O.FIREWALL, O.RESOURCES
FTP_ITC.1/CM	O.LOAD

Table 8 SFRs and Security Objectives

7.3.3 DEPENDENCIES

7.3.3.1 SFRs DEPENDENCIES

Requirements	CC Dependencies	Satisfied Dependencies	
FDP_ACC.2/FIREWALL	(FDP_ACF.1)	FDP_ACF.1/FIREWALL	
FDP_ACF.1/FIREWALL	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.2/FIREWALL, FMT_MSA.3/FIREWALL	
FDP_IFC.1/JCVM	(FDP_IFF.1)	FDP_IFF.1/JCVM	
FDP_IFF.1/JCVM	(FDP_IFC.1) and (FMT_MSA.3)	FDP_IFC.1/JCVM, FMT_MSA.3/JCVM	
FDP_RIP.1/OBJECTS	No Dependencies		
FMT_MSA.1/JCRE	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/FIREWALL, FMT_SMR.1	
FMT_MSA.1/JCVM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_SMF.1, FMT_SMR.1	
FMT_MSA.2/FIREWALL_JCVM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_MSA.1) and (FMT_SMR.1)	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_SMR.1	
FMT_MSA.3/FIREWALL	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_SMR.1	
FMT_MSA.3/JCVM	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/JCVM, FMT_SMR.1	
FMT_SMF.1	No Dependencies		
FMT_SMR.1	(FIA_UID.1)	FIA_UID.2/AID	
FCS_CKM.1	(FCS_CKM.2 or FCS_CKM.5 or FCS_COP.1) and	FCS_COP.1, FCS_RNG.1 FCS_CKM.6	

	(FCS_RBG.1 or FCS_RNG.1) and (FCS_CKM.6)		
FCS_CKM.6	(FCS_CKM.1 or FCS_CKM.5 or FDP_ITC.1 or FDP_ITC.2)	FCS_CKM.1	
FCS_COP.1	(FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1 or FCS_CKM.5) and (FCS_CKM.6)	FCS_CKM.1, FCS_CKM.6	
FCS_RNG.1	No Dependencies		
FDP_RIP.1/ABORT	No Dependencies		
FDP_RIP.1/APDU	No Dependencies		
FDP_RIP.1/bArray	No Dependencies		
FDP_RIP.1/GlobalArray	No Dependencies		
FDP_RIP.1/KEYS	No Dependencies		
FDP_RIP.1/TRANSIENT	No Dependencies		
FDP_ROL.1/FIREWALL	(FDP_ACC.1 or FDP_IFC.1)	FDP_ACC.2/FIREWALL, FDP_IFC.1/JCVM	
FAU_ARP.1	(FAU_SAA.1)		
FDP_SDI.2/DATA	No Dependencies		
FPR_UNO.1	No Dependencies		
FPT_FLS.1	No Dependencies		
FPT_TDC.1	No Dependencies		
FIA_ATD.1/AID	No Dependencies		
FIA_UID.2/AID	No Dependencies		
FIA_USB.1/AID	(FIA_ATD.1)	FIA_ATD.1/AID	

FMT_MTD.1/JCRE	(FMT_SMF.1) and (FMT_SMR.1)	FMT_SMF.1, FMT_SMR.1	
FMT_MTD.3/JCRE	(FMT_MTD.1)	FMT_MTD.1/JCRE	
FDP_ITC.2/Installer	(FDP_ACC.1 or FDP_IFC.1) and (FPT_TDC.1) and (FTP_ITC.1 or FTP_TRP.1)	FDP_IFC.2/CM, FTP_ITC.1/CM, FPT_TDC.1	
FMT_SMR.1/Installer	(FIA_UID.1)		
FPT_FLS.1/Installer	No Dependencies		
FPT_RCV.3/Installer	(AGD_OPE.1)	AGD_OPE.1	
FDP_ACC.2/ADEL	(FDP_ACF.1)	FDP_ACF.1/ADEL	
FDP_ACF.1/ADEL	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.2/ADEL, FMT_MSA.3/ADEL	
FDP_RIP.1/ADEL	No Dependencies		
FMT_MSA.1/ADEL	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/ADEL, FMT_SMF.1/ADEL, FMT_SMR.1/ADEL	
FMT_MSA.3/ADEL	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/ADEL, FMT_SMR.1/ADEL	
FMT_SMF.1/ADEL	No Dependencies		
FMT_SMR.1/ADEL	(FIA_UID.1)		
FPT_FLS.1/ADEL	No Dependencies		
FDP_RIP.1/ODEL	No Dependencies		
FPT_FLS.1/ODEL	No Dependencies		
FCO_NRO.2/CM	(FIA_UID.1)	FIA_UID.1/CM	
FDP_IFC.2/CM	(FDP_IFF.1)	FDP_IFF.1/CM	
FDP_IFF.1/CM	(FDP_IFC.1) and (FMT_MSA.3)	FDP_IFC.2/CM, FMT_MSA.3/CM	
FDP_UIT.1/CM	(FDP_ACC.1 or FDP_IFC.1) and	FDP_IFC.2/CM, FTP_ITC.1/CM	

	(FTP_ITC.1 or FTP_TRP.1)		
FIA_UID.1/CM	No Dependencies		
FMT_MSA.1/CM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_IFC.2/CM, FMT_SMF.1/CM, FMT_SMR.1/CM	
FMT_MSA.3/CM	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/CM, FMT_SMR.1/CM	
FMT_SMF.1/CM	No Dependencies		
FMT_SMR.1/CM	(FIA_UID.1)	FIA_UID.1/CM	
FTP_ITC.1/CM	No Dependencies		

Table 9 SFRs Dependencies

7.3.3.1.1 RATIONALE FOR THE EXCLUSION OF DEPENDENCIES

The dependency FIA_UID.1 of FMT_SMR.1/Installer is discarded. This PP does not require the identification of the "installer" since it can be considered as part of the TSF.

The dependency FIA_UID.1 of FMT_SMR.1/ADEL is discarded. This PP does not require the identification of the "deletion manager" since it can be considered as part of the TSF.

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

The dependency FAU_SAA.1 of FAU_ARP.1 is discarded. The dependency of FAU_ARP.1 on FAU_SAA.1 assumes that a "potential security violation" generates an audit event. On the contrary, the events listed in FAU_ARP.1 are self-contained (arithmetic exception, ill-formed bytecodes, access failure) and ask for a straightforward reaction of the TSFs on their occurrence at runtime. The JCVM or other components of the TOE detect these events during their usual working order. Thus, there is no mandatory audit recording in this PP.

7.3.3.2 SARS DEPENDENCIES

Requirements	CC Dependencies	Satisfied Dependencies
ADV_ARC.1	(ADV_FSP.1) and (ADV_TDS.1)	ADV_FSP.4, ADV_TDS.3
ADV_FSP.4	(ADV_TDS.1)	ADV_TDS.3
ADV_IMP.1	(ADV_TDS.3) and (ALC_TAT.1)	ADV_TDS.3, ALC_TAT.1
ADV_TDS.3	(ADV_FSP.4)	ADV_FSP.4

AGD_OPE.1	(ADV_FSP.1)	ADV_FSP.4
AGD_PRE.1	No Dependencies	
ALC_CMC.4	(ALC_CMS.1) and (ALC_DVS.1) and (ALC_LCD.1)	ALC_CMS.4, ALC_DVS.2, ALC_LCD.1
ALC_CMS.4	No Dependencies	
ALC_DEL.1	No Dependencies	
ALC_DVS.2	No Dependencies	
ALC_FLR.2	No Dependencies	
ALC_LCD.1	No Dependencies	
ALC_TAT.1	(ADV_IMP.1)	ADV_IMP.1
ASE_CCL.1	(ASE_ECD.1) and (ASE_INT.1) and (ASE_REQ.1)	ASE_ECD.1, ASE_INT.1, ASE_REQ.2
ASE_ECD.1	No Dependencies	
ASE_INT.1	No Dependencies	
ASE_OBJ.2	(ASE_SPD.1)	ASE_SPD.1
ASE_REQ.2	(ASE_ECD.1) and (ASE_OBJ.2)	ASE_ECD.1, ASE_OBJ.2
ASE_SPD.1	No Dependencies	
ASE_TSS.1	(ADV_FSP.1) and (ASE_INT.1) and (ASE_REQ.1)	ADV_FSP.4, ASE_INT.1, ASE_REQ.2
ATE_COV.2	(ADV_FSP.2) and (ATE_FUN.1)	ADV_FSP.4, ATE_FUN.1
ATE_DPT.1	(ADV_ARC.1) and (ADV_TDS.2) and (ATE_FUN.1)	ADV_ARC.1, ADV_TDS.3, ATE_FUN.1
ATE_FUN.1	(ATE_COV.1)	ATE_COV.2
ATE_IND.2	(ADV_FSP.2) and (AGD_OPE.1) and (AGD_PRE.1) and (ATE_COV.1) and (ATE_FUN.1)	ADV_FSP.4, AGD_OPE.1, AGD_PRE.1, ATE_COV.2, ATE_FUN.1
AVA_VAN.5	(ADV_ARC.1) and (ADV_FSP.4) and (ADV_IMP.1) and (ADV_TDS.3) and (AGD_OPE.1) and (AGD_PRE.1) and (ATE_DPT.1)	ADV_ARC.1, ADV_FSP.4, ADV_IMP.1, ADV_TDS.3, AGD_OPE.1, AGD_PRE.1, ATE_DPT.1

Table 10 SARs Dependencies

7.3.4 RATIONALE FOR THE SECURITY ASSURANCE REQUIREMENTS

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

7.3.5 ALC_DVS.2 SUFFICIENCY OF SECURITY MEASURES

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

7.3.6 AVA_VAN.5 ADVANCED METHODICAL VULNERABILITY ANALYSIS

The TOE is intended to operate in hostile environments. AVA_VAN.5 "Advanced methodical vulnerability analysis" is considered as the expected level for Java Card technology-based products hosting sensitive applications, in particular for payment and identity areas. AVA_VAN.5 has dependencies on ADV_ARC.1, ADV_FSP.4, ADV_TDS.3, ADV_IMP.1, AGD_OPE.1,AGD_PRE.1 and ATE_DPT.1. All of them are satisfied by EAL4.

7.3.7 ALC_FLR.2 FLAW REPORTING PROCEDURES

Due to the nature of the TOE and embedding product, it is necessary to provide flaw reporting procedures to track all reported security flaws in each release of the TOE.

In order for the developer (of Java Card technology-based product) to be able to act appropriately upon security flaw reports from TOE users, and to know to whom to send corrective fixes, TOE users need to understand how to submit security flaw reports to the developer. Flaw remediation guidance from the developer to the TOE user is necessary to ensure that TOE users are aware of this important information.

ALC_FLR.2 has no dependencies.

APPENDIX 1: JAVA CARD SYSTEM – OPEN CONFIGURATION AUGMENTATION PACKAGES

1. OVERVIEW

This Appendix introduces the security elements specific to the optional features in Java Card specifications, as Augmentation Packages. This concerns the biometric templates management, the JCRMI, the extended memory, the Sensitive Result, Sensitive Array, Monotonic Counter, Cryptographic Certificate Management, Key Derivation and System Time packages described hereafter:

1. The Biometric templates package if integrated into the TOE shall be securely managed. *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.*
2. JCRMI package provides a mechanism for a client application running on the CAD platform to invoke a method on a remote object on the card. The CAD issues commands to the card, which in turn dispatches them to the appropriate object. The applet owner of those objects controls the access to exported objects and the JCRE ensures coherence and synchronization of the remote object with its on-card representative.
3. The Extended Memory package is an API-based mechanism to access the external memory outside the addressable Java Card VM space.
4. The SensitiveArrays package provides methods for creating and handling integrity-sensitive array objects.
5. SensitiveResult package provides methods for asserting results of sensitive functions.
6. MonotonicCounter package provides methods for creating a counter that can only be increased, never decreased.
7. The Cryptographic Certificate Management package provides secure management of public key certificates from the javacardx.security.cert package of the Java Card platform.
8. The Key Derivation Functions package provides classes implementing cryptographic derivation functions.
9. The System Time package provides methods for handling system time, suitable for timestamps or for estimating intervals between events.

What follows are security elements related to these optional packages introduced in Java Card specifications. Therefore, it is the responsibility of the Security Target editor to include these security elements if the package is supported.

Application note:
For instance, the ST writer shall indicate whether JCRMI is implemented in the TOE and whether it is activated or not. If the TOE provides JCRMI functionality, the full range of SFRs applies. Otherwise, the ST writer shall ignore JCRMI dedicated threats, objectives and requirements. This applies to all the other optional packages presented above.

2. PACKAGE BIOMETRIC TEMPLATES

Package Biometric templates (short name – Biometrics) is a functional package that provides the functionality to securely manage biometric templates as described below. It has no dependencies on other functional packages. The component rationale for selecting this package is to provide the functionality to securely manage biometric templates.

SECURITY PROBLEM DEFINITION & SECURITY OBJECTIVES

There is one additional asset:

D.BIO

Any biometric template.

To be protected from unauthorized disclosure and modification.

Application note:
This asset is similar to D.PIN asset. This means that all the attacks that threaten the PIN code shall threaten the Biometric template and therefore the same Security Objectives and SFRs apply.

There is one additional security objective:

O.BIO-MNGT

The TOE shall provide a means to securely manage biometric templates. This concerns the optional packages javacardx.biometry or javacardx.biometry1toN of the Java Card platform

These objectives cover the following threats:

Security Objectives	Threats
O.BIO-MNGT	T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA

Security Objectives and Threats – Coverage

SECURITY REQUIREMENTS RATIONALE

O.BIO-MNGT This objective is ensured by FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FPR_UNO.1, FDP_ROL.1/FIREWALL and FDP_SDI.2/DATA security functional requirements. The applets that manage biometric templates rely on the security functions that implement these SFRs. The firewall security functions (FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL) shall protect the access to private and internal data of the templates. Note that the objective applies only to configurations including the javacardx.biometry or javacardx.biometry1toN packages defined in [JCAPI3].

The following table shows the relationship between SFRs and the security objective:

Security Objectives	Security Functional Requirements	Rationale
O.BIO-MNGT	FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FPR_UNO.1, FDP_ROL.1/FIREWALL, FDP_SDI.2/DATA, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL	Appendix 1.2

Security Objectives and SFRs - Coverage

3. PACKAGE JCRMI

Package JCRMI (short name – JCRMI) is a functional package that defines a mechanism for a client application running on the CAD platform to invoke a method on a remote object on the card. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for a client application running on the CAD platform to invoke a method on a remote object on the card.

SECURITY PROBLEM DEFINITION & SECURITY OBJECTIVE

The following security threat is introduced:

T.EXE-CODE-REMOTE

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

Directly threatened asset(s): D.APP_CODE.

Application Note:

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

This threat applies only if the TOE provides JCRMI functionality.

The following security objective covers the threat above:

O.REMOTE

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

Application Note:

This objective applies only if the TOE provides JCRMI functionality.

SECURITY OBJECTIVE RATIONALE

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

This objective covers the following threat:

Security Objectives	Threats
O.REMOTE	T.EXE-CODE-REMOTE

Security Objectives and Threats – Coverage

SECURITY FUNCTIONAL REQUIREMENTS

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

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

Object	Description
O.REMOTE_MTHD	A method of a remote interface
O.REMOTE_OBJ	A remote object is an instance of a class that implements one (or more) remote interfaces. The remote interface can extend, directly or indirectly, the interface java.rmi.Remote ([JCAPI3])..
O.RMI_SERVICE	These are instances of the class javacardx.rmi.RMIService. They are the objects that actually process the RMI services.
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.

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

Information	Description
I.RORD	Remote object reference descriptors which provide information concerning: (i) the identification of the remote object and (ii) the implementation class of the object or the interfaces implemented by the class of the object. The descriptor is the only object's information to which the CAD can access.

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

Security attribute	Description/Value
Class	Identifies the implementation class of the remote object.
Identifier	The Identifier of a remote object or method is a number that uniquely identifies the remote object or method, respectively.
ExportedInfo	Boolean (indicates whether the remote object is exportable or not).
Remote	An object is Remote if it is an instance of a class that directly or indirectly implements the interface java.rmi.Remote. It applies only if the TOE provides JCRMI functionality.
Returned References	The set of remote object references that have been sent to the CAD during the applet selection session. This attribute is implementation dependent.

(*) 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.

Operation	Description
OP.GET_ROR(O.APPLET,...)	This operation retrieves the initial remote object reference of a RMI based applet. This reference is the seed which the CAD client application needs to begin remote method invocations.
OP.INVOKE(O.RMI_SERVICE,...)	This operation requests a remote method invocation on the remote object
OP.RET_RORD(S.JCRE,S.CAD,I.RORD)	Send a remote object reference descriptor to the CAD.

FDP_ACC.2/JCRMI Complete access control

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

Refinement:

The operations involved in this policy are:

- OP.GET_ROR,
- OP.INVOKE.

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

FDP_ACF.1/JCRMI Security attributes based access control

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

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

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

- **R.JAVA.18: S.CAD may perform OP.GET_ROR upon O.APPLET only if O.APPLET is the currently selected applet, and there exists an O.RMI_SERVICE with a registered initial reference to an O.REMOTE_OBJ that is owned by O.APPLET.**
- **R.JAVA.19: S.JCRE may perform OP.INVOKE upon O.RMI_SERVICE, O.ROR and O.REMOTE_MTHD only if O.ROR is valid (as defined in [JCRE3], §8.5) and it belongs to the Returned References of O.RMI_SERVICE, and if the Identifier of O.REMOTE_MTHD matches one of the remote methods in the Class of the O.REMOTE_OBJ to which O.ROR makes reference**.

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

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

any subject but S.JCRE to O.REMOTE_OBJ and O.REMOTE_MTHD for the purpose of performing a remote method invocation.

Application Note:

FDP_ACF.1.2/JCRMI:

- The validity of a remote object reference is specified as a lifetime characterization. The security attributes involved in the rules for determining valid remote object references are the Returned References of the O.RMI_SERVICE and the Active Applets (see FMT_REV.1.1/JCRMI and FMT_REV.1.2/JCRMI). The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([JCRE3], §8.5.2 and [JCAPI3]).

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

FDP_IFC.1/JCRMI Subset information flow control

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

Application Note:

FDP_IFC.1.1/JCRMI:

- Array parameters of remote method invocations must be allocated on the card as global arrays objects. References to global arrays cannot be stored in class variables, instance variables or array components. The control of the flow of that kind of information has already been specified in FDP_IFC.1.1/JCVM.
- A remote object reference descriptor is sent from the card to the CAD either as the result of a successful applet selection command ([JCRE3], §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 ([JCRE3],§8.3.5.1).

FDP_IFF.1/JCRMI Simple security attributes

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

Subjects/Information	Security attributes
I.RORD	ExportedInfo

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

OP.RET_RORD(S.JCRE, S.CAD, I.RORD) is permitted only if the attribute ExportedInfo of I.RORD has the value "true" ([JCRE3], §8.5).

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

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

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

Application Note:

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

FMT_MSA.1/EXPORT Management of security attributes

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

Application Note:

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

FMT_MSA.1/REM_REFS Management of security attributes

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

FMT_MSA.3/JCRMI Static attribute initialisation

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

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

Application Note:

FMT_MSA.3.1/JCRMI:

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

FMT_MSA.3.2/JCRMI:

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

FMT_REV.1/JCRMI Revocation

FMT_REV.1.1/JCRMI The TSF shall restrict the ability to revoke **the Returned References associated with the object O.RMI_SERVICE** under the control of the TSF to **the Java Card RE**.

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

Application Note:

The rules are described in [JCRE3], §8.5

FMT_SMF.1/JCRMI Specification of Management Functions

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

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

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

FMT_SMR.1/JCRMI Security roles

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

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

Application Note:

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

SECURITY REQUIREMENTS RATIONALE

In addition to the SFRs covering already the security objectives listed below, if the TOE supports the JCRMI functionality, the additional SFRs contributes in covering these security objectives as follows:

O.SID This objective is also met by the following SFRs: FMT_MSA.3/JCRMI, FMT_SMF.1/JCRMI. Lastly, installation procedures ensure protection against forgery (the AID of an applet is under the control of the TSFs) or re-use of identities (FIA_UID.2/AID, FIA_USB.1/AID).

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

O.GLOBAL_ARRAYS_CONFID This objective is also met by the following SFRs: protection of the array parameters of remotely invoked methods, which are global as well, is covered by the general initialization of method parameters (FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/ABORT, FDP_RIP.1/KEYS, FDP_RIP.1/ADEL and FDP_RIP.1/TRANSIENT).

O.RESOURCES Memory management is controlled by the TSF (FMT_SMR.1/JCRMI, and FMT_SMF.1/JCRMI).

O.REMOTE The access to the TOE's internal data and the flow of information from the card to the CAD required by the JCRMI service is under control of the JCRMI access control policy (FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI) and the JCRMI information flow control policy (FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI). The security functional requirements of the class FMT

(FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI and FMT_SMR.1/JCRMI) included in the group RMIG also contribute to meet this objective.

RATIONALE TABLES OF SECURITY OBJECTIVES AND SFRS

Security Objectives	Security Functional Requirements	Rationale
O.SID	FIA_ATD.1/AID, FIA_UID.2/AID, FMT_MSA.1/JCRE, FMT_MSA.3/JCRMI, FMT_MSA.1/REM_REFS, FMT_MSA.1/EXPORT, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_MSA.3/FIREWALL, FMT_MSA.1/CM, FMT_MSA.3/CM, FDP_ITC.2/Installer, FMT_SMF.1/CM, FMT_SMF.1/ADEL, FMT_SMF.1/JCRMI, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FIA_USB.1/AID, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM	Appendix 1 section 3
O.FIREWALL	FDP_IFC.1/JCVM, FDP_IFF.1/JCVM, FMT_SMR.1/Installer, FMT_MSA.1/CM, FMT_MSA.3/CM, FMT_SMR.1/CM, FMT_MSA.3/FIREWALL, FMT_SMR.1, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_SMR.1/ADEL, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI, FMT_MSA.1/JCRE, FDP_ITC.2/Installer, FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, FDP_ACC.2/FIREWALL, FDP_ACF.1/FIREWALL, FMT_SMF.1/ADEL, FMT_SMF.1/JCRMI, FMT_SMF.1/CM, FMT_SMF.1, FMT_MSA.2/FIREWALL_JCVM, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_MSA.1/JCVM, FMT_MSA.3/JCVM	Appendix 1 section 3
O.RESOURCES	FAU_ARP.1, FDP_ROL.1/FIREWALL, FMT_SMR.1/Installer, FMT_SMR.1, FMT_SMR.1/ADEL, FMT_SMR.1/JCRMI, FPT_FLS.1/Installer, FPT_FLS.1/ODEL, FPT_FLS.1, FPT_FLS.1/ADEL, FPT_RCV.3/Installer, FMT_SMR.1/CM, FMT_SMF.1/ADEL, FMT_SMF.1/JCRMI, FMT_SMF.1/CM, FMT_SMF.1, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE	Appendix 1 section 3
O.REMOTE	FDP_ACC.2/JCRMI, FDP_ACF.1/JCRMI, FDP_IFC.1/JCRMI, FDP_IFF.1/JCRMI, FMT_MSA.1/EXPORT, FMT_MSA.1/REM_REFS, FMT_MSA.3/JCRMI, FMT_REV.1/JCRMI, FMT_SMR.1/JCRMI	Appendix 1 section 3

Security Objectives and SFRs - Coverage

Security Functional Requirements	Security Objectives
FDP_ACC.2/JCRMI	O.FIREWALL, O.REMOTE
FDP_ACF.1/JCRMI	O.FIREWALL, O.REMOTE
FDP_IFC.1/JCRMI	O.REMOTE
FDP_IFF.1/JCRMI	O.REMOTE
FMT_MSA.1/EXPORT	O.SID, O.FIREWALL, O.REMOTE
FMT_MSA.1/REM_REFS	O.SID, O.FIREWALL, O.REMOTE
FMT_MSA.3/JCRMI	O.SID, O.FIREWALL, O.REMOTE
FMT_REV.1/JCRMI	O.FIREWALL, O.REMOTE
FMT_SMF.1/JCRMI	O.SID, O.FIREWALL, O.RESOURCES
FMT_SMR.1/JCRMI	O.FIREWALL, O.RESOURCES, O.REMOTE

SFRs and Security Objectives

Requirements	CC Dependencies	Satisfied Dependencies
FDP_ACC.2/JCRMI	(FDP_ACF.1)	FDP_ACF.1/JCRMI
FDP_ACF.1/JCRMI	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.2/JCRMI, FMT_MSA.3/JCRMI
FDP_IFC.1/JCRMI	(FDP_IFF.1)	FDP_IFF.1/JCRMI
FDP_IFF.1/JCRMI	(FDP_IFC.1) and (FMT_MSA.3)	FDP_IFC.1/JCRMI, FMT_MSA.3/JCRMI
FMT_MSA.1/EXPORT	(FDP_ACC.1 or FDP_IFC.1), (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/JCRMI, FMT_SMF.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.1/REM_REFS	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.2/JCRMI, FMT_SMF.1/JCRMI, FMT_SMR.1/JCRMI
FMT_MSA.3/JCRMI	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/REM_REFS, FMT_SMR.1/JCRMI
FMT_REV.1/JCRMI	(FMT_SMR.1)	FMT_SMR.1/JCRMI
FMT_SMF.1/JCRMI	No dependencies	
FMT_SMR.1/JCRMI	(FIA_UID.1)	

SFRs dependencies

RATIONALE FOR THE EXCLUSION OF DEPENDENCIES

The dependency FIA_UID.1 of FMT_SMR.1/JCRMI is discarded. This PP does not require the identification of the "applet" since it can be considered as part of the TSF.

4. PACKAGE EXTENDED MEMORY

Package Extended Memory (short name – ExtMem) is a functional package that defines a mechanism to access external memory outside the addressable Java Card VM space. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism to access external memory outside the addressable Java Card VM space.

SECURITY OBJECTIVE

One new security objective is introduced:

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 objective covers the following threats:

Security Objectives	Threats
O.EXT-MEM	T.CONFID-JCS-CODE, T.CONFID-APPLI-DATA, T.INTEG-APPLI-DATA T.CONFID-JCS-DATA, T.INTEG-APPLI-CODE, T.INTEG-JCS-CODE, T.INTEG-JCS-DATA

Security Objectives and Threats – Coverage

T.CONFID-JCS-CODE This threat is countered by the security objective for extended memory (O.EXT-MEM) such that the memory that contains Java Card System Code, is not addressable from the external memory and therefore it's confidentiality is protected.

T.INTEG-JCS-CODE This threat is countered by the security objective for extended memory (O.EXT-MEM) such that the memory that contains Java Card System Code, is not addressable from the external memory and therefore it's integrity is protected.

T.INTEG-APPLI-CODE This threat is countered by the security objective for extended memory (O.EXT-MEM) such that the memory that contains Application Code, is not addressable from the external memory and therefore it's integrity is protected.

T.CONFID-APPLI-DATA This threat is countered by the security objective for extended memory (O.EXT-MEM) such that Java Card System memory, which contains Java Card Application Data, is not addressable from the external memory.

T.INTEG-APPLI-DATA This threat is countered by the security objective for extended memory (O.EXT-MEM) such that Java Card System memory, which contains Application Data, is not addressable from the external memory.

T.CONFID-JCS-DATA This threat is countered by the security objective for extended memory (O.EXT-MEM) such that Java Card System memory, which contains Java Card System Data, is not addressable from the external memory.

T.INTEG-JCS-DATA This threat is countered by the security objective for extended memory (O.EXT-MEM) such that Java Card System memory, which contains Java Card System Data, is not addressable by the external memory.

SECURITY FUNCTIONAL REQUIREMENTS

Thi group of SFRs contains the following security requirements for the management of the external memory, introduced in the version 2.2.2 of the Java Card System (cf. [JCAPI3], optional package javacardx.external).

The External Memory access policy relies on the following additional objects, operations and security attributes.

Object/Operation/Attribute	Description
O.EXT_MEM_INSTANCE	Any External Memory Instance created from the MemoryAccess Interface of the Java Card API [JCAPI3].
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.
Address space	Accessible memory portion.

FDP_ACC.1/EXT_MEM 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.**

FDP_ACF.1/EXT_MEM Security attribute based access control

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
O.EXT_MEM_INSTANCE	Address space.

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:

- R.JAVA.20: 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.
- R.JAVA.21: 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.22: 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.

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

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

Application note:

The actual mechanism for creating an instance of external memory is implementation-dependent. This rule only states that the accessible address space must not interfere with that of the Java Card System.

The creation and the access to an external memory instance fall in the scope of the Firewall rules.

FMT_MSA.1/EXT_MEM 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 attributes **address space** to **the Java Card RE**.

FMT_MSA.3/EXT_MEM Static attribute initialisation

FMT_MSA.3.1/EXT_MEM The TSF shall enforce the **EXTERNAL MEMORY 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/EXT_MEM Specification of Management Functions

FMT_SMF.1.1/EXT_MEM The TSF shall be capable of performing the following management functions: **set up the address space security attribute**.

FMT_SMR.1/EXT_MEM Security roles

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

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

SECURITY REQUIREMENTS RATIONALE

O.SID Subjects' identity is AID-based (applets, packages and CAP files), and is met by the following SFRs: FDP_ITC.2/Installer, FIA_ATD.1/AID, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM, FMT_MSA.1/REM_REFS, FMT_MSA.1/EXPORT, FMT_MSA.1/ADEL, FMT_MSA.1/CM, FMT_MSA.3/ADEL, FMT_MSA.3/FIREWALL, FMT_MSA.3/JCVM, FMT_MSA.3/CM, FMT_SMF.1/CM, FMT_SMF.1/ADEL, FMT_SMF.1/ADEL, FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_SMF.1/EXT_MEM, FMT_MSA.1/EXT_MEM and FMT_MSA.3/EXT_MEM.

Lastly, installation procedures ensure protection against forgery (the AID of an applet is under the control of the TSFs) or re-use of identities (FIA_UID.2/AID, FIA_USB.1/AID).

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

O.FIREWALL This objective is met by the FIREWALL access control policy FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL, the JCVM information flow control policy

(FDP_IFF.1/JCVM, FDP_IFC.1/JCVM) and the functional requirement FDP_ITC.2/Installer. The functional requirements of the class FMT (FMT_MTD.1/JCRE, FMT_MTD.3/JCRE, FMT_SMR.1/Installer, FMT_SMR.1, FMT_SMF.1, FMT_SMR.1/ADEL, FMT_SMF.1/ADEL, FMT_SMF.1/CM, FMT_SMF.1/EXT_MEM, FMT_MSA.1/EXT_MEM, FMT_MSA.3/EXT_MEM, FMT_SMR.1/EXT_MEM, FMT_MSA.1/CM, FMT_MSA.3/CM, FMT_SMR.1/CM, FMT_MSA.2/FIREWALL_JCVM, FMT_MSA.3/FIREWALL, FMT_MSA.3/JCVM, FMT_MSA.1/ADEL, FMT_MSA.3/ADEL, FMT_MSA.1/JCRE, FMT_MSA.1/JCVM) also indirectly contribute to meet this objective.

O.EXT-MEM The Java Card System memory is protected against applet's attempts of unauthorized access through the external memory facilities by the EXTERNAL MEMORY access control policy (FDP_ACC.1/EXT_MEM, FDP_ACF.1/EXT_MEM), which first controls the accessible address space, then controls the effective read and write operations. External memory management is controlled by the TSF (FMT_SMF.1/EXT_MEM).

The following tables show the relationship between SFRs and objectives and the SFR dependencies.

Security Objectives	Security Functional Requirements	Rationale
O.SID	FMT_SMF.1/EXT_MEM, FMT_MSA.1/EXT_MEM, FMT_MSA.3/EXT_MEM	Appendix 1 section 4
O.RESOURCES	FMT_SMF.1/EXT_MEM, FMT_SMR.1/EXT_MEM,	Appendix 1 section 4
O.FIREWALL	FMT_SMF.1/EXT_MEM, FMT_MSA.1/EXT_MEM, FMT_MSA.3/EXT_MEM, FMT_SMR.1/EXT_MEM,	Appendix 1 section 4
O.EXT-MEM	FDP_ACC.1/EXT_MEM, FDP_ACF.1/EXT_MEM, FMT_SMF.1/EXT_MEM	Appendix 1 section 4

Security Objectives and SFRs - Coverage

Security Functional Requirements	Security Objectives
FDP_ACC.1/EXT_MEM	O.EXT-MEM
FDP_ACF.1/EXT_MEM	O.EXT-MEM
FMT_MSA.1/EXT_MEM	O.SID, O.FIREWALL
FMT_MSA.3/EXT_MEM	O.SID, O.FIREWALL
FMT_SMF.1/EXT_MEM	O.SID, O.RESOURCES, O.FIREWALL, O.EXT-MEM
FMT_SMR.1/EXT_MEM,	O.RESOURCES, O.FIREWALL,

SFRs and Security Objectives

Requirements	CC Dependencies	Satisfied Dependencies
FDP_ACC.1/EXT_MEM	(FDP_ACF.1)	FDP_ACF.1/EXT_MEM
FDP_ACF.1/EXT_MEM	(FDP_ACC.1) and (FMT_MSA.3)	FDP_ACC.1/EXT_MEM, FMT_MSA.3/EXT_MEM
FMT_MSA.1/EXT_MEM	(FDP_ACC.1 or FDP_IFC.1) and (FMT_SMF.1) and (FMT_SMR.1)	FDP_ACC.1/EXT_MEM, FMT_SMF.1/EXT_MEM
FMT_MSA.3/EXT_MEM	(FMT_MSA.1) and (FMT_SMR.1)	FMT_MSA.1/EXT_MEM,
FMT_SMF.1/EXT_MEM	No dependencies	
FMT_SMR.1/EXT_MEM	(FIA_UID.1)	

SFRs dependencies

RATIONALE FOR THE EXCLUSION OF DEPENDENCIES

The dependency FIA_UID.1 of FMT_SMR.1/EXT_MEM is discarded. This PP does not require the identification of the "Java Card RE" since it can be considered as part of the TSF.

5. PACKAGE SENSITIVE ARRAY

Package SensitiveArrays (short name – SensitiveArrays) is a functional package that defines a mechanism for creating and handling integrity-sensitive array objects. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for creating and handling integrity-sensitive array objects.

#.INTEG-APPLI-DATA-PHYS Integrity-sensitive application data must be protected against unauthorized modification by physical attacks.

T.PHYSICAL

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

This threatens all the identified assets.

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

Application note:

If sensitive array is supported by the TOE, this threat covers the following sub-threat exploiting specifically the listed assets below:

- The attacker performs a physical manipulation to alter (part of) an application's integrity-sensitive data. See #.INTEG-APPLI-DATA-PHYS for details.

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

SECURITY OBJECTIVE

O.SENSITIVE_ARRAYS_INTEG

The TOE shall ensure that only the currently selected applications may have a write access to the integrity-sensitive array object (javacard.framework.SensitiveArrays) created by that application. Any unauthorized modification through physical attacks to that integrity-sensitive array must be detected by the TOE and notified to the application.

These objectives cover the following threats:

Security Objectives	Threats
O.SENSITIVE_ARRAYS_INTEG	T.PHYSICAL

Security Objectives and Threats – Coverage

SECURITY OBJECTIVE RATIONALE

T.PHYSICAL If the sensitive array is supported by the TOE, this threat is partially covered by the security objective O.SENSITIVE_ARRAYS_INTEG which requires the TOE to detect and notify the application if any unauthorized modification of the integrity-sensitive array object through physical attacks occurred.

SECURITY FUNCTIONAL REQUIREMENTS

FDP_SDI.2/ARRAY Integrity_Sensitive_Array

FDP_SDI.2.1/ARRAY The TSF shall monitor user data stored in containers controlled by the TSF for **integrity errors** on all objects, based on the following attributes: **user data stored in arrays created by the makeIntegritySensitiveArray() method of the javacard.framework.SensitiveArrays class.**

FDP_SDI.2.2/ARRAY Upon detection of a data integrity error, the TSF shall **throw an exception**.

Application Note:

This requirement applies in particular to the arrays created by the makeIntegritySensitiveArray() method of the javacard.framework.SensitiveArrays class (if supported).

These objectives are covered by the following SFRs:

Security Objectives	SFRs

O_SENSITIVE_ARRAYS_INTEG	FDP_SDI_2/ARRAY

Security Objectives and SFRs – Coverage

SECURITY REQUIREMENTS RATIONALE

O.SENSITIVE_ARRAYS_INTEG This security objective is covered directly by the SFR FDP_SDI.2/ARRAY Integrity_Senstive_Array which ensures that integrity errors related to the user data stored in sensitive arrays are detected by the TOE.

Requirements	CC Dependencies	Satisfied Dependencies
FDP_SDI.2/ARRAY	No dependencies	

SFRs dependencies

6. PACKAGE SENSITIVE RESULT

Package SensitiveResult (short name – SensitiveResult) is a functional package that defines a mechanism for asserting results of sensitive functions. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for asserting results of sensitive functions.

SECURITY PROBLEM DEFINITION

#.INTEG-APPLI-DATA-PHYS Integrity-sensitive application data must be protected against unauthorized modification by physical attacks.

T.PHYSICAL

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

This threatens all the identified assets.

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

Application note:

If sensitive result is supported by the TOE, this threat covers the following sub-threat exploiting specifically the listed assets below:

- The attacker performs a physical manipulation to alter (part of) an application's integrity-sensitive data. See #.INTEG-APPLI-DATA-PHYS for details.

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

SECURITY OBJECTIVE

O.SENSITIVE_RESULTS_INTEG

The TOE shall ensure that the sensitive results (javacardx.security.SensitiveResults) of sensitive operations executed by applications through the Java Card API are protected in integrity specifically against physical attacks.

These objectives cover the following threats:

Security Objectives	Threats
O.SENSITIVE_RESULTS_INTEG	T.PHYSICAL

Security Objectives and Threats – Coverage

SECURITY OBJECTIVE RATIONALE

T.PHYSICAL If the sensitive result is supported by the TOE, this threat is partially covered by the security objective O.SENSITIVE_RESULTS_INTEG which ensures that sensitive results are protected against unauthorized modification by physical attacks.

SECURITY FUNCTIONAL REQUIREMENTS

FDP_SDI.2/RESULT Integrity_Sensitive_Result

FDP_SDI.2.1/RESULT The TSF shall monitor user data stored in containers controlled by the TSF for **[assignment: integrity errors]** on all objects, based on the following attributes: **[assignment: sensitive API result stored in the javacardx.security.SensitiveResult class].**

FDP_SDI.2.2/RESULT Upon detection of a data integrity error, the TSF shall **[assignment: throw an exception]**.

Application Note:

This requirement applies in particular to the results stored by the javacardx.security.SensitiveResult class (if supported).

These objectives are covered by the following SFRs:

Security Objectives	SFRs
O.SENSITIVE_RESULTS_INTEG	**FDP_SDI.2/RESULT Integrity_Sensitive_Result**

Security Objectives and SFRs – Coverage

SECURITY REQUIREMENTS RATIONALE

O.SENSITIVE_RESULTS_INTEG This security objective is covered directly by the SFR FDP_SDI.2/RESULT Integrity_Senstive_Result which ensures that integrity errors related to the senstive API result are detected by the TOE.

Requirements	CC Dependencies	Satisfied Dependencies
FDP_SDI_2/ RESULT	No dependencies	

SFRs dependencies

7. PACKAGE MONOTONIC COUNTERS

Package MonotonicCounter (short name – MonotonicCounter) is a functional package that defines a mechanism for creating a counter that can only be increased. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for creating a counter that can only be increased.

SECURITY PROBLEM DEFINITION

There are no additional assets.

Application Note:

The scope of D.APP_I_DATA is widened in view of the Monotonic counters functionality, i.e. the asset described in section 5.1.1 now also address and cover monotonic counters.

There is one additional security objective:

O.MTC-CTR-MNGT

The TOE shall provide a means to securely manage value of the monotonic counter. This concerns the optional package javacardx.security.util of the Java Card platform.

This objective cover the following threats:

Security Objectives	Threats
O.MTC-CTR-MNGT	T.INTEG-APPLI-DATA

T.INTEG-APPLI-DATA This threat is countered by the security objective for monotonic counter management (O.MTC-CTR-MNGT) such that value of the monotonic counter will be protected against any unauthorized change.

SECURITY FUNCTIONAL REQUIREMENTS

FDP_SDI.2/MONOTONIC_COUNTER

FDP_SDI.2.1/MONOTONIC_COUNTER The TSF shall monitor user data stored in containers controlled by the TSF for **integrity errors** on all objects, based on the following attributes: **stored user data i.e. the counter value in the MonotonicCounter object.**

FDP_SDI.2.2/MONOTONIC_COUNTER Upon detection of a data integrity error, the TSF shall **throw an exception**.

Application Note:

This requirement applies to MonotonicCounter objects created by the getInstance() method of the javacardx.security.util.MonotonicCounter class (if supported).

SECURITY REQUIREMENTS RATIONALE

O.MTC-CTR-MNGT This security objective is ensured by FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT, FDP_ROL.1/FIREWALL and FDP_SDI.2/MONOTONIC_COUNTER security functional requirements. The TSFs behind these are implemented by API classes. The firewall security functions FDP_ACC.2/FIREWALL and FDP_ACF.1/FIREWALL shall protect the access to private and internal data of the objects. Note that the objective applies only to configurations including the javacardx.security.util package defined in [JCAPI3].

The following table shows the relationship between SFRs and the security objective:

Security Objectives	Security Functional Requirements
O.MTC-CTR-MNGT	FDP_RIP.1/ODEL, FDP_RIP.1/OBJECTS, FDP_RIP.1/APDU, FDP_RIP.1/GlobalArray FDP_RIP.1/bArray, FDP_RIP.1/ABORT, FDP_RIP.1/ADEL, FDP_RIP.1/TRANSIENT, FDP_ROL.1/FIREWALL, FDP_SDI.2/MONOTONIC_COUNTER

Security Objectives and SFRs – Coverage

Requirements	CC Dependencies	Satisfied Dependencies
FDP_SDI_2/MONOTONIC_COUNTER	No dependencies	

SFRs dependencies

8. PACKAGE CRYPTOGRAPHIC CERTIFICATE MANAGEMENT

Package Cryptographic Certificate Management (short name – CryptoCertMgmt) is a functional package that defines a mechanism for secure management of public key certificates. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for secure management of public key certificates.

SECURITY PROBLEM DEFINITION & SECURITY OBJECTIVES

There are no additional assets.

Application Note:
The scope of D.APP_I_DATA is widened in view of the Cryptographic Certificate Management functionality, i.e. the asset described in section 5.1.1 now also address and cover cryptographic certificates.

There is one additional security objective:

O.CRT-MNGT
The TOE shall provide a means to securely manage cryptographic certificates. This concerns the optional package javacardx.security.cert of the Java Card platform

These objectives cover the following threats:

Security Objectives	Threats
O.CRT-MNGT	T.INTEG-APPLI-DATA

Security Objectives and Threats – Coverage

T.INTEG-APPLI-DATA This threat is countered by the security objective for certificate management (O.CRT-MNGT) such that certificate data will be protected against any unauthorized change.

SECURITY FUNCTIONAL REQUIREMENTS

FDP_SDI.2/CRT_MNGT

FDP_SDI.2.1/CRT_MNGT The TSF shall monitor user data stored in containers controlled by the TSF for **integrity errors** on all objects, based on the following attributes: **cryptographic certificate.**

FDP_SDI.2.2/CRT_MNGT Upon detection of a data integrity error, the TSF shall **throw an exception**.

Application Note:

This requirement applies to Certificate objects if optional javacardx.security.cert package is supported.

FCS_COP.1/CRT_MNGT Certificate Management

FCS_COP.1.1/CRT_MNGT The TSF shall perform **[assignment: verification of X.509 Certificate]** 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:

Refer to Appendix 3 to define the allowed/available algorithms as per Java Card API specifications [JCAPI3] The ST Author should choose the asymmetric cryptography algorithm combined with a hashing algorithm to perform verification of certificate signatures. For each algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm.

- The TOE shall provide a subset of asymmetric cryptography algorithms combined with hashing algorithms defined in [JCAPI3] (see javacardx.crypto.Cipher and javacardx.security packages).

SECURITY REQUIREMENTS RATIONALE

O.CRT-MNGT This objective is ensured by FDP_SDI.2/CRT_MNGT and FCS_COP.1/CRT_MNGT security functional requirements. The applets that manage cryptographic certificates rely on the security functions that implement these SFRs. Note that the objective applies only to configurations including the javacardx.security.cert package defined in [JCAPI3].

The following table shows the relationship between SFRs and the security objective:

Security Objectives	Security Functional Requirements
O.CRT-MNGT	FDP_SDI.2/CRT_MNGT, FCS_COP.1/CRT_MNGT

Security Objectives and SFRs – Coverage

Requirements	CC Dependencies	Satisfied Dependencies
FDP_SDI_2/CRT_MNGT	No dependencies	
FCS_COP.1/CRT_MNGT	(FCS_CKM.1 or FCS_CKM.5 or FDP_ITC.1 or FDP_ITC.2) and (FCS_CKM.6)	FCS_CKM.1, FCS_CKM.6

SFRs dependencies

9. PACKAGE KEY DERIVATION FUNCTIONS (KDF)

Package Key Derivation (short name – KDF) is a functional package that defines classes implementing cryptographic derivation functions. It has no dependencies on other functional packages. The component rationale for selecting this package is to define classes implementing cryptographic derivation functions.

SECURITY PROBLEM DEFINITION

ASSETS:

No additional assets are introduced. Assets to be protected are D.APP_KEYs, D.API_DATA, D.CRYPTO, D.JCS_CODE, D.JCS_DATA and D.SEC_DATA.

THREATS:

No additional threats are introduced if KDF API is supported by the TOE. Specific threats against KDF refer to security aspects #.KEY-MNGT, #.EXE-JCS-CODE, #.INTEG-JCS-CODE, #.INTEG-JCS-DATA and #.SCP.

SECURITY OBJECTIVES

Security objectives O.CIPHER and O.KEY-MNGT are relevant security objectives if KDF API extension package is implemented. There are no additional security objectives.

SECURITY FUNCTIONAL REQUIREMENTS

FCS_CKM.5/KDF Cryptographic key derivation

FCS_CKM.5.1/KDF The TSF shall derive cryptographic keys **[assignment: key type]** from **[assignment: input parameters]** in accordance with a specified key derivation algorithm **[assignment: key derivation algorithm]** and specified cryptographic key sizes **[assignment: list of key sizes]** that meet the following: **[assignment: list of standards]**.

Application Note:

This requirement applies if optional javacardx.security.derivation package is supported:
If javacardx.security.derivation package is supported, security objectives O.OPERATE, O.RNG and O.KEY-MNGT are further covered by the following SFRs:

Security Objectives	SFRs

O.CIPHER, O.KEY-MNGT	FCS_CKM.5/KDF

Security Objectives and SFRs – Coverage

SECURITY REQUIREMENTS RATIONALE

O.CIPHER This security objective is ensured by FCS_CK.5/KDF security functional requirement. The TSFs behind these are implemented by API classes. Note that the objective applies only to configurations including the javacardx.security.derivation package defined in [JCAPI3].

O.KEY-MNGT This security objective is ensured by FCS_CK.5/KDF security functional requirement. The TSFs behind these are implemented by API classes. Note that the objective applies only to configurations including the javacardx.security.derivation package defined in [JCAPI3].

Requirements	CC Dependencies	Satisfied Dependencies
FCS_CKM.5/KDF	(FCS_CKM.2 or FCS_COP.1) and (FCS_CKM.6)	FCS_COP.1, FCS_CKM.6

SFRs dependencies

10. PACKAGE SYSTEM TIME

Package System Time (short name – SysTime) is a functional package that defines a mechanism for handling system time, suitable for timestamps or for estimating intervals between events. It has no dependencies on other functional packages. The component rationale for selecting this package is to define a mechanism for handling system time, suitable for timestamps or for estimating intervals between events.

SECURITY PROBLEM DEFINITION

No additional assets are introduced. System time is part of D.JCS_DATA therefore the asset to be protected is D.JCS_DATA.

THREATS:

No additional threats are introduced if system time API is supported by the TOE. Specific threats against System Time refer to security aspect #.INTEG-JCS-DATA.

SECURITY OBJECTIVES

Security objectives O.OPERATE, O.RESOURCE are relevant security objectives if system time API extension package is implemented. There are no additional security objectives.

SECURITY FUNCTIONAL REQUIREMENTS

FPT_STM.1/SYS_TIME

FPT_STM.1.1/SYS_TIME The TSF shall be able to provide reliable time stamps.

Application Note:

This requirement applies if optional javacardx.framework.time package is supported.

If javacardx.framework.time package is supported, security objectives O.OPERATE, O.RESOURCE are further covered by the following SFRs:

Security Objectives	SFRs
O.OPERATE, O.RESOURCE	FPT_STM.1.1/SYS_TIME

Security Objectives and SFRs – Coverage

SECURITY REQUIREMENTS RATIONALE

Refer to section 7.4 for security requirement rationales for of O.OPERATE and O.RESOURCE and their mapping to relevant SFRs. **O.OPERATE, and O.RESOURCE** security objectives are further covered the SFR FPT_STM.1/SYS_TIME.

Requirements	CC Dependencies	Satisfied Dependencies
FPT_STM.1/SYS_TIME	No dependencies	

APPENDIX 2: A UNIFIED VIEW OF CONFIGURATIONS

This section provides an all-embracing presentation of the security environment, security objectives and functional requirements of the JCS configurations. The tables below do not only make explicit the contents proper to each configuration but also reflect 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 and version 3 Classic Edition of the Java Card platform. Those corresponding to TSF data are: **D.JCS_CODE, D.JCS_DATA, D.SEC_DATA, D.API_DATA, and D.CRYPTO.**

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

Assumption	*Closed*	*Open 2.2.x and 3.0.5 Classic Edition*	*Open 3.1 and later*
A.NO-INSTALL	X		
A.NO-DELETION	X		
A.CAP_FILE		X	X
A.DELETION		X	X
A.VERIFICATION	X	X	X

Table A3-1: Assumptions by configuration

The threats to the assets against which specific protection is required within the configurations or their environments are displayed in Table A3-2. The post-issuance installation of applets introduces one threat (T.INSTALL), and two more (T.INTEG-APPLI-CODE.LOAD T.INTEG-APPLI-DATA.LOAD) since bytecode verification is performed off-card.

Threat	*Closed*	*Open 2.2.x and 3.0.5 Classic Edition*	*Open 3.1 and later*
T.PHYSICAL	X	X	X
T.CONFID-JCS-CODE	X	X	X
T.CONFID-APPLI-DATA	X	X	X
T.CONFID-JCS-DATA	X	X	X
T.INTEG-APPLI-CODE	X	X	X
T.INTEG-JCS-CODE	X	X	X
T.INTEG-APPLI-DATA	X	X	X

T.INTEG-JCS-DATA	X	X	X
T.SID.1	X	X	X
T.SID.2	X	X	X
T.EXE-CODE.1	X	X	X
T.EXE-CODE.2	X	X	X
T.NATIVE	X	X	X
T.RESOURCES	X	X	X
T.INTEG-APPLI-CODE.LOAD		X	X
T.INTEG-APPLI-DATA.LOAD		X	X
T.INSTALL		X	X
T.EXE-CODE-REMOTE		**X (Open 2.2.2 optional feature)**	**X (Open 3.1 optional feature)**
T.DELETION		X	X
T.OBJ-DELETION		X	X

Table A3-2: Threats by configuration

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

TOE security objective	*Closed*	*Open 2.2.x and 3.0.5 Classic Edition*	*Open 3.1 and later*
O.SID	X	X	X
O.OPERATE	X	X	X
O.RESOURCES	X	X	X
O.FIREWALL	X	X	X
O.NATIVE	X	X	X
O.REALLOCATION	X	X	X
O.GLOBAL_ARRAYS_CONFID	X	X	X
O.GLOBAL_ARRAYS_INTEG	X	X	X
O.ARRAY_VIEWS_CONFID	X		X
O.ARRAY_VIEWS_INTEG	X		X
O.ALARM	X	X	X
O.TRANSACTION	X	X	X
O.CIPHER	X	X	X
O.RNG	X	X	X
O.PIN-MNGT	X	X	X
O.KEY-MNGT	X	X	X
O.INSTALL		X	X
O.LOAD		X	X
O.DELETION		X	X

O.OBJ-DELETION		**X**	**X**
O.REMOTE		**X (Open 2.2.2 optional feature)**	**X (Open 3.1 optional feature)**
O.BIO-MNGT	**X (Closed 2.2.2 optional feature)**	**X (Open 2.2.2 optional feature)**	**X (Open 3.1 optional feature)**
O.EXT-MEM	**X (Closed 2.2.2 optional feature)**	**X (Open 2.2.2 optional feature)**	**X (Open 3.1 optional feature)**
O.SENSITIVE_ARRAYS_INTEG		**X (Open 3.0.5) optional feature)**	**X (Open 3.1 optional feature)**
O.SENSITIVE_RESULTS_INTEG		**X (Open 3.0.5) optional feature)**	**X (Open 3.1 optional feature)**
O.MTC-CTR-MNGT			**X (Open 3.1 optional feature)**
O.CRT-MNGT			**X (Open 3.1 optional feature)**

Table A3-3: TOE Security objectives by configuration

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

Environment security objective	*Closed*	*Open 2.2.x and 3.0.5 Classic Edition*	*Open 3.1 and later*
OE.SCP.RECOVERY	**X**	**X**	**X**
OE.SCP.SUPPORT	**X**	**X**	**X**
OE.SCP.IC	**X**	**X**	**X**
OE.NO-DELETION	**X**		
OE.NO-INSTALL	**X**		
OE.VERIFICATION	**X**	**X**	**X**
OE.CODE-EVIDENCE	**X**	**X**	**X**
OE.**CAP_FILE**		**X**	**X**
OE.CARD-MANAGEMENT	**X**	**X**	**X**

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

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

SFR	*Group*	*Closed*	*Open 2.2.x and 3Classic Edition(to 3.0.4)*	*Open 3.0.5*	*Open 3.1 and later*
FAU_ARP.1	*CoreG*	X	X	X	X
FCS_CKM.1	*CoreG*	X	X	X	X
FCS_CKM.6	*CoreG*	X	X	X	X
FCS_COP.1	*CoreG*	X	X	X	X
FDP_ACC.2/ FIREWALL	*CoreG*	X	X	X	X
FDP_ACF.1/ FIREWALL	*CoreG*	X	X	X	X
FDP_IFC.1/J CVM	*CoreG*	X	X	X	X
FDP_IFF.1/J CVM	*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/ TRANSIENT	*CoreG*	X	X	X	X
FDP_RIP.1/ OBJECTS	*CoreG*	X	X	X	X
FDP_ROL.1/ FIREWALL	*CoreG*	X	X	X	X
FDP_SDI.2/ DATA	*CoreG*	X	X	X	X
FIA_ATD.1/ AID	*CoreG*	X	X	X	X
FIA_UID.2/A ID	*CoreG*	X	X	X	X
FIA_USB.1/ AID	*CoreG*	X	X	X	X
FMT_MSA.1/ JCRE	*CoreG*	X	X	X	X
FMT_MSA.1/ JCVM	*CoreG*	X	X	X	X

FMT_MSA.2/ FIREWALL_J CVM	*CoreG*	**X**	**X**	**X**	**X**
FMT_MSA.3/ JCVM	*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 /JCRE	*CoreG*	**X**	**X**	**X**	**X**
FMT_SMR.1	*CoreG*	**X**	**X**	**X**	**X**
FMT_SMF.1	*CoreG*	**X**	**X**	**X**	**X**
FPR_UNO.1	*CoreG*	**X**	**X**	**X**	**X**
FPT_FLS.1	*CoreG*	**X**	**X**	**X**	**X**
FPT_TDC.1	*CoreG*	**X**	**X**	**X**	**X**
FDP_ITC.2/I nstaller	*InstG*		**X**	**X**	**X**
FMT_SMR.1/ Installer	*InstG*		**X**	**X**	**X**
FPT_FLS.1/I nstaller	*InstG*		**X**	**X**	**X**
FPT_RCV.3/ Installer	*InstG*		**X**	**X**	**X**
FDP_ACC.2/ ADEL	*ADELG*		**X**	**X**	**X**
FDP_ACF.1/ ADEL	*ADELG*		**X**	**X**	**X**
FMT_MSA.1/ ADEL	*ADELG*		**X**	**X**	**X**
FMT_MSA.3/ ADEL	*ADELG*		**X**	**X**	**X**
FMT_SMR.1/ ADEL	*ADELG*		**X**	**X**	**X**
FMT_SMF.1/ ADEL	*ADELG*		**X**	**X**	**X**
FDP_RIP.1/ ADEL	*ADELG*		**X**	**X**	**X**
FPT_FLS.1/A DEL	*ADELG*		**X**	**X**	**X**
FDP_ACC.2/ JCRMI	*RMIG*		**(X)**	**(X)**	**(X)**
FDP_ACF.1/ JCRMI	*RMIG*		**(X)**	**(X)**	**(X)**
FDP_IFC.1/J CRMI	*RMIG*		**(X)**	**(X)**	**(X)**

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

FMT_SMF.1/ EXT_MEM	*EMG*		**(X)**	**(X)**	**(X)**
FDP_SDI.2/ ARRAY	*ESEC*			**(X)**	**(X)**
FDP_SDI.2/ RESULT	*ESEC*			**(X)**	**(X)**
FDP_SDI.2/ MONOTONIC_COUNTER	*ESEC*				**(X)**
FDP_SDI.2/ CRT_MNGT	*ESEC*				**(X)**
FCS_COP.1/ CRT_MNGT	*ESEC*				**(X)**
FCS_CKM.5/ KDF	*ESEC*				**(X)**
FPT_STM.1/ SYS_TIME	*ESEC*				**(X)**

Table A3-5: Security Functional Requirements by configurations

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

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

Table A3-6: Configurations and roles

APPENDIX 3: SUPPORTED CRYPTOGRAPHIC ALGORITHMS

This Appendix lists all the cryptographic algorithms supported by Java Card specifications. The Security Target writer should check which cryptographic algorithms are supported by their respective certification scheme.

Javacardx.crypto.Cipher	introduced in JavaCard version	Description
ALG_DES_CBC_NOPAD	<=2.1	Cipher algorithm ALG_DES_CBC_NOPAD provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and does not pad input data.
ALG_DES_CBC_ISO9797_M1	<=2.1	Cipher algorithm ALG_DES_CBC_ISO9797_M1 provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 1 scheme.
ALG_DES_CBC_ISO9797_M2	<=2.1	Cipher algorithm ALG_DES_CBC_ISO9797_M2 provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
ALG_DES_CBC_PKCS5	<=2.1	Cipher algorithm ALG_DES_CBC_PKCS5 provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the PKCS#5 scheme.
ALG_DES_ECB_NOPAD	<=2.1	Cipher algorithm ALG_DES_ECB_NOPAD provides a cipher using DES in ECB mode, and does not pad input data.
ALG_DES_ECB_ISO9797_M1	<=2.1	Cipher algorithm ALG_DES_ECB_ISO9797_M1 provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 1 scheme.
ALG_DES_ECB_ISO9797_M2	<=2.1	Cipher algorithm ALG_DES_ECB_ISO9797_M2 provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
ALG_DES_ECB_PKCS5	<=2.1	Cipher algorithm ALG_DES_ECB_PKCS5 provides a cipher using DES in ECB mode, and pads input data according to the PKCS#5 scheme.
ALG_RSA_ISO14888	<=2.1	Deprecated
ALG_RSA_PKCS1	<=2.1	Cipher algorithm ALG_RSA_PKCS1 provides a cipher using RSA, and pads input data according to the PKCS#1 (v1.5) scheme.
ALG_RSA_ISO9796	<=2.1	Deprecated

ALG_RSA_NOPAD	2.1.1	Cipher algorithm ALG_RSA_NOPAD provides a cipher using RSA and does not pad input data.
ALG_AES_BLOCK_128_CBC_NOPAD	2.2.0	Cipher algorithm ALG_AES_BLOCK_128_CBC_NOPAD provides a cipher using AES with block size 128 in CBC mode and does not pad input data.
ALG_AES_BLOCK_128_ECB_NOPAD	2.2.0	Cipher algorithm ALG_AES_BLOCK_128_ECB_NOPAD provides a cipher using AES with block size 128 in ECB mode and does not pad input data.
ALG_RSA_PKCS1_OAEP	2.2.0	Cipher algorithm ALG_RSA_PKCS1_OAEP provides a cipher using RSA, and pads input data according to the PKCS#1-OAEP scheme (IEEE 1363-2000).
ALG_KOREAN_SEED_ECB_NOPAD	2.2.2	Cipher algorithm ALG_KOREAN_SEED_ECB_NOPAD provides a cipher using the Korean SEED algorithm specified in the Korean SEED Algorithm specification provided by KISA, Korea Information Security Agency in ECB mode and does not pad input data.
ALG_KOREAN_SEED_CBC_NOPAD	2.2.2	Cipher algorithm ALG_KOREAN_SEED_CBC_NOPAD provides a cipher using the Korean SEED algorithm specified in the Korean SEED Algorithm specification provided by KISA, Korea Information Security Agency in CBC mode and does not pad input data.
ALG_AES_BLOCK_192_CBC_NOPAD	3.0.1	Deprecated
ALG_AES_BLOCK_192_ECB_NOPAD	3.0.1	Deprecated
ALG_AES_BLOCK_256_CBC_NOPAD	3.0.1	Deprecated
ALG_AES_BLOCK_256_ECB_NOPAD	3.0.1	Deprecated
ALG_AES_CBC_ISO9797_M1	3.0.1	Cipher algorithm ALG_AES_CBC_ISO9797_M1 provides a cipher using AES with block size 128 in CBC mode, and pads input data according to the ISO 9797 method 1 scheme.
ALG_AES_CBC_ISO9797_M2	3.0.1	Cipher algorithm ALG_AES_CBC_ISO9797_M2 provides a cipher using AES with block size 128 in CBC mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
ALG_AES_CBC_PKCS5	3.0.1	Cipher algorithm ALG_AES_CBC_PKCS5 provides a cipher using AES with block size 128 in CBC mode, and pads input data according to the PKCS#5 scheme.
ALG_AES_ECB_ISO9797_M1	3.0.1	Cipher algorithm ALG_AES_ECB_ISO9797_M1 provides a cipher using AES with block size 128 in ECB mode, and pads input data according to the ISO 9797 method 1 scheme.

ALG_AES_ECB_ISO9797_M2	3.0.1	Cipher algorithm ALG_AES_ECB_ISO9797_M2 provides a cipher using AES with block size 128 in ECB mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
ALG_AES_ECB_PKCS5	3.0.1	Cipher algorithm ALG_AES_ECB_PKCS5 provides a cipher using AES with block size 128 in ECB mode, and pads input data according to the PKCS#5 scheme.
CIPHER_AES_CBC	3.0.4	The cipher algorithm provides a cipher using AES with block size 128 in CBC mode.
CIPHER_AES_ECB	3.0.4	The cipher algorithm provides a cipher using AES with block size 128 in ECB mode.
CIPHER_DES_CBC	3.0.4	The cipher algorithm provides a cipher using DES in CBC mode.
CIPHER_DES_ECB	3.0.4	The cipher algorithm provides a cipher using DES in ECB mode.
CIPHER_KOREAN_SEED_CBC	3.0.4	The cipher algorithm provides a cipher using KOREAN_SEED in CBC mode.
CIPHER_KOREAN_SEED_ECB	3.0.4	The cipher algorithm provides a cipher using KOREAN_SEED in ECB mode.
CIPHER_RSA	3.0.4	The cipher algorithm provides a cipher using RSA.
PAD_NULL	3.0.4	This constant indicates that there is no discrete padding algorithm.
PAD_NOPAD	3.0.4	This padding algorithm choice requires that the data length is a multiple of the cipher algorithm block size.
PAD_ISO9797_M1	3.0.4	This padding algorithm choice requests padding based on the ISO 9797 method 1 scheme.
PAD_ISO9797_M2	3.0.4	This padding algorithm choice requests padding based on the ISO 9797 method 2 scheme.
PAD_ISO9797_1_M1_ALG3	3.0.4	This padding algorithm choice requests padding based on the ISO9797-1 MAC algorithm 3 with method 1.
PAD_ISO9797_1_M2_ALG3	3.0.4	This padding algorithm choice requests padding based on the ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000).
PAD_PKCS5	3.0.4	This padding algorithm choice requests padding based on the PKCS #5 scheme.
PAD_PKCS1	3.0.4	This padding algorithm choice requests padding based on the PKCS v1.5 scheme.
PAD_PKCS1_PSS	3.0.4	This padding algorithm choice requests padding based on the PKCS#1-PSS scheme (IEEE 1363-2000) with a default salt length equal to the length of the message digest and with the message digest algorithm of MGF1 being the same as the one of the parameter messageDigestAlgorithm.
PAD_PKCS1_OAEP	3.0.4	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-

		2000) with SHA-1 as default message digest algorithm for both the scheme and MGF1.
PAD_ISO9796	3.0.4	This padding algorithm choice requests padding based on the ISO/IEC 9796-2 signature scheme 1, signature production function B.6 with a trailer field option 1 and as specified in EMV 3.0 and EMV 4.0
PAD_ISO9796_MR	3.0.4	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 1, signature production function B.6 and giving message recovery with a trailer field option 1 - (also EMV 3.0, EMV 4.0).
PAD_RFC2409	3.0.4	This padding algorithm choice requests padding based on the RFC 2409 scheme.
ALG_AES_CTR	3.0.5	Cipher algorithm ALG_AES_CTR provides a cipher using AES in counter (CTR) mode.
CIPHER_AES_CTR	3.0.5	Cipher algorithm CIPHER_AES_CTR provides a cipher using AES in counter (CTR) mode.
PAD_PKCS1_OAEP_SHA224	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA-224 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA256	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA256 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA384	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA384 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA512	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA512 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA3_224	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA3-224 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA3_256	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA3-256 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA3_384	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA3-384 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_OAEP_SHA3_512	3.0.5	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000) with SHA3-512 as default message digest algorithm for both the scheme and MGF1.

CIPHER_AES_CFB	3.1	Cipher Algorithm CIPHER_AES_CFB provides a cipher using AES in Cipher Feedback (CFB) mode.
CIPHER_AES_XTS	3.1	Cipher Algorithm CIPHER_AES_XTS provides a cipher using AES in XEX Tweakable Block Cipher with Ciphertext Stealing (XTS) mode as defined in IEEE Std 1619.
CIPHER_SM2	3.1	The CIPHER_SM2 provides a cipher using SM2 encryption as defined in GM/T 0003.4-2012 (Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves Part 4: Public Key Encryption Algorithm)
CIPHER_SM4_ECB	3.1	The CIPHER_SM4_ECB provides a cipher using SM4 block cipher algorithm in CBC mode with 128-bit input blocks
CIPHER_SM4_CBC	3.1	The CIPHER_SM4_CBC constant provides a cipher using SM4 block cipher algorithm in CBC mode with 128-bit input blocks
ALG_AES_CFB	3.1	Cipher Algorithm ALG_AES_CFB provides a cipher using AES in Cipher Feedback (CFB) mode.
ALG_AES_XTS	3.1	Cipher Algorithm ALG_AES_XTS provides a cipher using AES in XEX Tweakable Block Cipher with Ciphertext Stealing (XTS) mode as defined in IEEE Std 1619.
PAD_ISO9796_MR_SCHEME_2	3.1	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 2, signature production function B.6 and giving message recovery with a trailer field option 1.
PAD_ISO9796_MR_SCHEME_3	3.1	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 3, signature production function B.6 and giving message recovery with a trailer field option 1.
PAD_ISO9796_MR_SCHEME_1_OPTION_2	3.2	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 1, signature production function B.6 and giving message recovery with a trailer field option 2.
PAD_ISO9796_MR_SCHEME_2_OPTION_2	3.2	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 2, signature production function B.6 and giving message recovery with a trailer field option 2.
PAD_ISO9796_MR_SCHEME_3_OPTION_2	3.2	This padding algorithm choice requests padding specified by ISO/IEC 9796-2 for signature scheme 3, signature production function B.6 and giving message recovery with a trailer field option 2.
PAD_PKCS1_OAEP_EXT_PARAMETERS	3.2	This padding algorithm choice requests padding based on the PKCS#1-OAEP scheme (IEEE 1363-2000)) with SHA-1 as default message digest algorithm for both the scheme and MGF1.
PAD_PKCS1_PSS_EXT_PARAMETERS	3.2	This padding algorithm choice requests padding based on the PKCS#1-PSS scheme (IEEE 1363-2000) with a default salt length equal to the

		length of the message digest and with a default message digest algorithm for MGF1 being the same as the one of the parameter messageDigestAlgorithm.
javacardx.crypto.AEADCipher		
ALG_AES_CCM	3.0.5	Cipher algorithm ALG_AES_CCM provides a cipher using AES in Counter with CBC-MAC mode as specified in RFC 3610.
ALG_AES_GCM	3.0.5	Cipher algorithm ALG_AES_GCM provides a cipher using AES in Galois/Counter Mode as specified in NIST SP 800-38D, November 2007.
CIPHER_AES_CCM	3.0.5	The cipher algorithm provides a cipher using AES in Counter with CBC-MAC mode as specified in http://tools.ietf.org/html/rfc3610
CIPHER_AES_GCM	3.0.5	The cipher algorithm provides a cipher using AES Galois/Counter Mode as specified http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf">NIST SP 800-38D, November 2007
javacard.security.Signature	**introduced in JavaCard version**	**Description**
ALG_DES_MAC4_NOPAD	<=2.1	Signature algorithm ALG_DES_MAC4_NOPAD generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC8_NOPAD	<=2.1	Signature algorithm ALG_DES_MAC8_NOPAD generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC4_ISO9797_M1	<=2.1	Signature algorithm ALG_DES_MAC4_ISO9797_M1 generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC8_ISO9797_M1	<=2.1	Signature algorithm ALG_DES_MAC8_ISO9797_M1 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC4_ISO9797_M2	<=2.1	Signature algorithm ALG_DES_MAC4_ISO9797_M2 generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC8_ISO9797_M2	<=2.1	Signature algorithm ALG_DES_MAC8_ISO9797_M2 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC4_PKCS5	<=2.1	Signature algorithm ALG_DES_MAC4_PKCS5 generates a 4-byte MAC (most significant 4 bytes

		of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
ALG_DES_MAC8_PKCS5	<=2.1	Signature algorithm ALG_DES_MAC8_PKCS5 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
ALG_RSA_SHA_ISO9796	<=2.1	Signature algorithm ALG_RSA_SHA_ISO9796 generates a 20-byte SHA digest, pads the digest according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, and encrypts it using RSA.
ALG_RSA_SHA_PKCS1	<=2.1	Signature algorithm ALG_RSA_SHA_PKCS1 generates a 20-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_RSA_MD5_PKCS1	<=2.1	Signature algorithm ALG_RSA_MD5_PKCS1 generates a 16-byte MD5 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_RSA_RIPEMD160_ISO9796	<=2.1	Signature algorithm ALG_RSA_RIPEMD160_ISO9796 generates a 20-byte RIPE MD-160 digest, pads the digest according to the ISO 9796 scheme, and encrypts it using RSA.
ALG_RSA_RIPEMD160_PKCS1	<=2.1	Signature algorithm ALG_RSA_RIPEMD160_PKCS1 generates a 20-byte RIPE MD-160 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_DSA_SHA	<=2.1	Signature algorithm ALG_DSA_SHA generates a 20-byte SHA digest and signs/verifies the digests using DSA.
ALG_RSA_SHA_RFC2409	<=2.1	Signature algorithm ALG_RSA_SHA_RFC2409 generates a 20-byte SHA digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.
ALG_RSA_MD5_RFC2409	<=2.1	Signature algorithm ALG_RSA_MD5_RFC2409 generates a 16-byte MD5 digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.
ALG_ECDSA_SHA	2.2.0	Signature algorithm ALG_ECDSA_SHA generates a 20-byte SHA digest and signs/verifies the digest using ECDSA.
ALG_AES_MAC_128_NOPAD	2.2.0	Signature algorithm ALG_AES_MAC_128_NOPAD generates a 16-byte MAC using AES with blocksize 128 in CBC mode and does not pad input data.
ALG_DES_MAC4_ISO9797_1_M2_ALG3	2.2.0	Signature algorithm ALG_DES_MAC4_ISO9797_1_M2_ALG3 generates a 4-byte MAC using triple DES with a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input

		data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
ALG_DES_MAC8_ISO9797_1_M2_ALG3	2.2.0	Signature algorithm ALG_DES_MAC8_ISO9797_1_M2_ALG3 generates an 8-byte MAC using triple DES with a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
ALG_RSA_SHA_PKCS1_PSS	2.2.0	Signature algorithm ALG_RSA_SHA_PKCS1_PSS generates a 20-byte SHA-1 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_RSA_MD5_PKCS1_PSS	2.2.0	Signature algorithm ALG_RSA_MD5_PKCS1_PSS generates a 16-byte MD5 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_RSA_RIPEMD160_PKCS1_PSS	2.2.0	Signature algorithm ALG_RSA_RIPEMD160_PKCS1_PSS generates a 20-byte RIPE MD-160 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_HMAC_SHA1	2.2.2	HMAC message authentication algorithm ALG_HMAC_SHA1 This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA1 as the hashing algorithm.
ALG_HMAC_SHA_256	2.2.2	HMAC message authentication algorithm ALG_HMAC_SHA_256 This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-256 as the hashing algorithm.
ALG_HMAC_SHA_384	2.2.2	HMAC message authentication algorithm ALG_HMAC_SHA_384 This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-384 as the hashing algorithm.
ALG_HMAC_SHA_512	2.2.2	HMAC message authentication algorithm ALG_HMAC_SHA_512 This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-512 as the hashing algorithm.
ALG_HMAC_MD5	2.2.2	HMAC message authentication algorithm ALG_HMAC_MD5 This algorithm generates an HMAC following the steps found in RFC: 2104 using MD5 as the hashing algorithm.
ALG_HMAC_RIPEMD160	2.2.2	HMAC message authentication algorithm ALG_HMAC_RIPEMD160 This algorithm generates an HMAC following the steps found in RFC: 2104 using RIPEMD160 as the hashing algorithm.
ALG_RSA_SHA_ISO9796_MR	2.2.2	Signature algorithmALG_RSA_SHA_ISO9796_MR generates 20-byte SHA-1 digest, pads it according

		to the ISO9796-2 specification and encrypts using RSA.
ALG_RSA_RIPEMD160_ISO9796_MR	2.2.2	Signature algorithmALG_RSA_RIPEMD160_ISO9796_MR generates 20-byte RIPE MD-160 digest, pads it according to the ISO9796-2 specification and encrypts using RSA.
ALG_ECDSA_SHA_256	3.0.1	Signature algorithm ALG_ECDSA_SHA_256 generates a 32-byte SHA-256 digest and signs/verifies the digest using ECDSA with the curve defined in the ECKey parameters - such as the P-256 curve specified in the Digital Signature Standards specification[NIST FIPS PUB 186-2].
ALG_ECDSA_SHA_384	3.0.1	Signature algorithm ALG_ECDSA_SHA_384 generates a 48-byte SHA-384 digest and signs/verifies the digest using ECDSA with the curve defined in the ECKey parameters - such as the P-384 curve specified in the Digital Signature Standards specification[NIST FIPS PUB 186-2].
ALG_AES_MAC_192_NOPAD	3.0.1	Deprecated
ALG_AES_MAC_256_NOPAD	3.0.1	**Deprecated**
ALG_ECDSA_SHA_224	3.0.1	Signature algorithm ALG_ECDSA_SHA_224 generates a 28-byte SHA-224 digest and signs/verifies the digest using ECDSA with the curve defined in the ECKey parameters - such as the P-224 curve specified in the Digital Signature Standards specification[NIST FIPS PUB 186-2].
ALG_ECDSA_SHA_512	3.0.1	Signature algorithm ALG_ECDSA_SHA_512 generates a 64-byte SHA-512 digest and signs/verifies the digest using ECDSA with the curve defined in the ECKey parameters - such as the P-521 curve specified in the Digital Signature Standards specification[NIST FIPS PUB 186-2].
ALG_RSA_SHA_224_PKCS1	3.0.1	Signature algorithm ALG_RSA_SHA_224_PKCS1 generates a 28-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_RSA_SHA_256_PKCS1	3.0.1	Signature algorithm ALG_RSA_SHA_256_PKCS1 generates a 32-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_RSA_SHA_384_PKCS1	3.0.1	Signature algorithm ALG_RSA_SHA_384_PKCS1 generates a 48-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
ALG_RSA_SHA_512_PKCS1	3.0.1	Signature algorithm ALG_RSA_SHA_512_PKCS1 generates a 64-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

ALG_RSA_SHA_224_PKCS1_PSS	3.0.1	Signature algorithm ALG_RSA_SHA_224_PKCS1_PSS generates a 28-byte SHA-224 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_RSA_SHA_256_PKCS1_PSS	3.0.1	Signature algorithm ALG_RSA_SHA_256_PKCS1_PSS generates a 32-byte SHA-256 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_RSA_SHA_384_PKCS1_PSS	3.0.1	Signature algorithm ALG_RSA_SHA_384_PKCS1_PSS generates a 48-byte SHA-384 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_RSA_SHA_512_PKCS1_PSS	3.0.1	Signature algorithm ALG_RSA_SHA_512_PKCS1_PSS generates a 64-byte SHA-512 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
ALG_DES_MAC4_ISO9797_1_M1_ALG3	3.0.4	Signature algorithm ALG_DES_MAC4_ISO9797_1_M1_ALG3 generates a 4-byte MAC using triple DES with a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 1, where input data is padded using method 1 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
ALG_DES_MAC8_ISO9797_1_M1_ALG3	3.0.4	Signature algorithm ALG_DES_MAC8_ISO9797_1_M1_ALG3 generates an 8-byte MAC using triple DES with a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 1, where input data is padded using method 1 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
SIG_CIPHER_DES_MAC4	3.0.4	The signature algorithm generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
SIG_CIPHER_DES_MAC8	3.0.4	The signature algorithm generates a 8-byte MAC (most significant 8 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. The signature algorithm generates a 8-byte MAC (most significant 8 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
SIG_CIPHER_RSA	3.0.4	The signature algorithm uses the RSA cipher.
SIG_CIPHER_DSA	3.0.4	The signature algorithm uses the DSA cipher. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order: SEQUENCE ::= { r INTEGER, s INTEGER }

SIG_CIPHER_ECDSA	3.0.4	The signature algorithm uses the ECDSA cipher. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order: SEQUENCE ::= { r INTEGER, s INTEGER }
SIG_CIPHER_AES_MAC128	3.0.4	The signature algorithm generates a 16-byte MAC using AES with block size 128 in CBC mode.
SIG_CIPHER_HMAC	3.0.4	The signature algorithm generates an HMAC following the steps found in RFC: 2104 using the specified hashing algorithm.
SIG_CIPHER_KOREAN_SEED_MAC	3.0.4	The signature algorithm generates a 16-byte MAC using Korean SEED in CBC mode.
ALG_AES_CMAC_128	3.0.5	Signature algorithm ALG_AES_CMAC_128 generates a 16-byte Cipher-based MAC (CMAC) using AES with blocksize 128 in CBC mode with ISO9797_M2 padding scheme.
SIG_CIPHER_ECDSA_PLAIN	3.0.5	The signature algorithm uses the ECDSA cipher. The signature is encoded as an octet string R || S as specified in section 5.2.1 of BSI TR-03111.
SIG_CIPHER_AES_CMAC128	3.0.5	The signature algorithm generates an HMAC following the steps found in RFC: 2104 using the specified hashing algorithm.
ALG_KOREAN_SEED_MAC_NOPAD	3.0.5 == ALG_SEED_MAC_NOPAD 2.2.2	Signature algorithm ALG_KOREAN_SEED_MAC_NOPAD generates an 16-byte MAC using Korean SEED in CBC mode.
SIG_CIPHER_EDDSA	3.1	Parameter used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pure EdDSA) with curves defined in RFC 8032
SIG_CIPHER_EDDSAPH	3.1	Used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pre-hash EdDSA) with curves defined in RFC 8032
SIG_CIPHER_SM2	3.1	Used to generate elliptic curve signature scheme using SM2 curve as defined in GM/T 0003.2-2012 (Public Key Cryptographic Algorithm SM2 based on Elliptic Curves Part 2: Digital Signature Algorithm).
SIG_CIPHER_EDDSA_ED25519	3.2	Used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pure EdDSA) for the variant ed25519 defined in RFC 8032 (Note: there is no context in this case).
SIG_CIPHER_EDDSA_ED448	3.2	Used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pure EdDSA) with an empty context and for the variant ed448 defined in RFC 8032.

SIG_CIPHER_EDDSAPH_ED25519	3.2	Used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pre-hash EdDSA) with an empty context and for the variant ed25519ph defined in RFC 8032.
SIG_CIPHER_EDDSAPH_ED448	3.2	Used to generate elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (pre-hash EdDSA) with an empty context and for the variant ed448 defined in RFC 8032.
javacard.security.MessageDigest	**introduced in JavaCard version**	**Description**
ALG_SHA	<=2.1	Message Digest algorithm SHA-1.
ALG_MD5	<=2.1	Message Digest algorithm MD5.
ALG_RIPEMD160	<=2.1	Message Digest algorithm RIPE MD-160.
ALG_SHA_256	2.2.2	Message Digest algorithm SHA-256.
ALG_SHA_384	2.2.2	Message Digest algorithm SHA-384.
ALG_SHA_512	2.2.2	Message Digest algorithm SHA-512.
ALG_SHA_224	3.0.1	Message Digest algorithm SHA-224.
ALG_SHA3_224	3.0.5	Message Digest algorithm SHA3-224.
ALG_SHA3_256	3.0.5	Message Digest algorithm SHA3-256.
ALG_SHA3_384	3.0.5	Message Digest algorithm SHA3-384.
ALG_SHA3_512	3.0.5	Message Digest algorithm SHA3-512.
ALG_SM3	3.1	Message Digest algorithm SM3.
javacard.security.RandomData	**introduced in JavaCard version**	**Description**
ALG_PSEUDO_RANDOM	<=2.1	**Deprecated.** As of release 3.0.5.
ALG_SECURE_RANDOM	<=2.1	**Deprecated.** As of release 3.0.5.
ALG_FAST	3.0.5	Utility random number generation algorithm.
ALG_KEYGENERATION	3.0.5	This algorithm creates random numbers suitable to be used for key and nonce generation.
ALG_PRESEEDED_DRBG	3.0.5	Deterministic Random Bit Generator (DRBG) algorithm.
ALG_TRNG	3.0.5	True Random Number Generation (TRNG) algorithm.
javacard.security.KeyBuilder	**introduced in JavaCard version**	**Description**
TYPE_DES_TRANSIENT_RESET	<=2.1	Key object which implements interface type DESKey with CLEAR_ON_RESET transient key data.
TYPE_DES_TRANSIENT_DESELECT	<=2.1	Key object which implements interface type DESKey with CLEAR_ON_DESELECT transient key data.
TYPE_DES LENGTH_DES	<=2.1	Key object which implements interface type DESKey with persistent key data. DES Key Length = 64.

TYPE_DES LENGTH_DES3_2KEY	<=2.1	Key object which implements interface type DESKey with persistent key data. DES Key Length = 128.
TYPE_DES LENGTH_DES3_3KEY	<=2.1	Key object which implements interface type DESKey with persistent key data. DES Key Length = 192.
TYPE_AES_TRANSIENT_RESET	2.2.0	Key object which implements interface type AESKey with CLEAR_ON_RESET transient key data.
TYPE_AES_TRANSIENT_DESELECT	2.2.0	Key object which implements interface type AESKey with CLEAR_ON_DESELECT transient key data.
TYPE_AES LENGTH_AES_128	2.2.0	Key object which implements interface type AESKey with persistent key data. AES Key Length = 128.
TYPE_AES LENGTH_AES_192	2.2.0	Key object which implements interface type AESKey with persistent key data. AES Key Length = 193.
TYPE_AES LENGTH_AES_256	2.2.0	Key object which implements interface type AESKey with persistent key data. AES Key Length = 256.
TYPE_AES LENGTH_AES_512	3.1	Key object which implements interface type AESKey with persistent key data. AES Key Length = 512.
TYPE_RSA_PUBLIC LENGTH_RSA_512	<=2.1	Key object which implements interface type RSAPublicKey. RSA Key Length = 512.
TYPE_RSA_PUBLIC LENGTH_RSA_736	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 736.
TYPE_RSA_PUBLIC LENGTH_RSA_768	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 768.
TYPE_RSA_PUBLIC LENGTH_RSA_896	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 896.
TYPE_RSA_PUBLIC LENGTH_RSA_1024	<=2.1	Key object which implements interface type RSAPublicKey. RSA Key Length = 1024.
TYPE_RSA_PUBLIC LENGTH_RSA_1280	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 1280.
TYPE_RSA_PUBLIC LENGTH_RSA_1536	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 1536.
TYPE_RSA_PUBLIC LENGTH_RSA_1984	2.2.0	Key object which implements interface type RSAPublicKey. RSA Key Length = 1984.
TYPE_RSA_PUBLIC LENGTH_RSA_2048	<=2.1	Key object which implements interface type RSAPublicKey. RSA Key Length = 2048.
TYPE_RSA_PUBLIC LENGTH_RSA_3072	3.0.5	Key object which implements interface type RSAPublicKey. RSA Key Length = 3072.
TYPE_RSA_PUBLIC LENGTH_RSA_4096	3.0.1	Key object which implements interface type RSAPublicKey. RSA Key Length = 4096.
TYPE_RSA_PRIVATE LENGTH_RSA_512	<=2.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 512.

TYPE_RSA_PRIVATE LENGTH_RSA_736	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 736.
TYPE_RSA_PRIVATE LENGTH_RSA_768	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 768.
TYPE_RSA_PRIVATE LENGTH_RSA_896	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 896.
TYPE_RSA_PRIVATE LENGTH_RSA_1024	<=2.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 1024.
TYPE_RSA_PRIVATE LENGTH_RSA_1280	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 1280.
TYPE_RSA_PRIVATE LENGTH_RSA_1536	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 1536.
TYPE_RSA_PRIVATE LENGTH_RSA_1984	2.2.0	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 1984.
TYPE_RSA_PRIVATE LENGTH_RSA_2048	<=2.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 2048.
TYPE_RSA_PRIVATE LENGTH_RSA_3072	3.0.5	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 3072.
TYPE_RSA_PRIVATE LENGTH_RSA_4096	3.0.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form. RSA Key Length = 4096.
TYPE_RSA_PRIVATE_TRANSIENT_RESET	3.0.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form, with CLEAR_ON_RESET transient key data.
TYPE_RSA_PRIVATE_TRANSIENT_DESELECT	3.0.1	Key object which implements interface type RSAPrivateKey which uses modulus/exponent form, with CLEAR_ON_DESELECT transient key data.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_512	<=2.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 512.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_736	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 736.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_768	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 768.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_896	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 896.

TYPE_RSA_CRT_PRIVATE LENGTH_RSA_1024	<=2.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 1024.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_1280	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 1280.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_1536	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 1536.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_1984	2.2.0	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 1984.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_2048	<=2.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 2048.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_3072	3.0.5	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 3072.
TYPE_RSA_CRT_PRIVATE LENGTH_RSA_4096	3.0.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem. RSA Key Length = 4096.
TYPE_RSA_CRT_PRIVATE_TRANSIENT_RESET	3.0.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem, with CLEAR_ON_RESET transient key data.
TYPE_RSA_CRT_PRIVATE_TRANSIENT_DESELECT	3.0.1	Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem, with CLEAR_ON_DESELECT transient key data.
TYPE_DSA_PRIVATE LENGTH_DSA_512	<=2.1	Algorithmic key type ALG_TYPE_DSA_PRIVATE choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method. DSA Key Length = 512.
TYPE_DSA_PRIVATE LENGTH_DSA_768	<=2.1	Algorithmic key type ALG_TYPE_DSA_PRIVATE choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method. DSA Key Length = 768.
TYPE_DSA_PRIVATE LENGTH_DSA_1024	<=2.1	Algorithmic key type ALG_TYPE_DSA_PRIVATE choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method. DSA Key Length = 1024.
TYPE_DSA_PRIVATE_TRANSIENT_RESET	3.0.1	Key object which implements the interface type DSAPrivateKey for the DSA algorithm, with CLEAR_ON_RESET transient key data.
TYPE_DSA_PRIVATE_TRANSIENT_DESELECT	3.0.1	Key object which implements the interface type DSAPrivateKey for the DSA algorithm, with CLEAR_ON_DESELECT transient key data.
TYPE_DSA_PUBLIC LENGTH_DSA_512	<=2.1	Key object which implements the interface type DSAPublicKey for the DSA algorithm. DSA Key Length = 512.

TYPE_DSA_PUBLIC LENGTH_DSA_768	<=2.1	Key object which implements the interface type DSAPublicKey for the DSA algorithm. DSA Key Length = 768.
TYPE_DSA_PUBLIC LENGTH_DSA_1024	<=2.1	Key object which implements the interface type DSAPublicKey for the DSA algorithm. DSA Key Length = 1024
TYPE_DSA_PARAMETERS	3.1	Key object containing domain parameters which implements interface types DSAKey and Key
TYPE_EC_F2M_PRIVATE LENGTH_EC_F2M_113	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 113.
TYPE_EC_F2M_PRIVATE LENGTH_EC_F2M_131	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 131.
TYPE_EC_F2M_PRIVATE LENGTH_EC_F2M_163	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 163.
TYPE_EC_F2M_PRIVATE LENGTH_EC_F2M_193	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 193.
TYPE_EC_F2M_PRIVATE_TRANSIENT_RESET	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis, with CLEAR_ON_RESET transient key data.
TYPE_EC_F2M_PRIVATE_TRANSIENT_DESELECT	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis, with CLEAR_ON_DESELECT transient key data.
TYPE_EC_F2M_PARAMETERS	3.1	Key object containing domain parameters which implements interface types ECKey and Key
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_112	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 112.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_128	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 128.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_160	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 160.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_192	2.2.0	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 192.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_224	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 224.

TYPE_EC_FP_PRIVATE LENGTH_EC_FP_256	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 256.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_384	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 384.
TYPE_EC_FP_PRIVATE LENGTH_EC_FP_521	3.0.4	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields. EC Key Length = 521.
TYPE_EC_FP_PRIVATE_TRANSIENT_RESET	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields, with CLEAR_ON_RESET transient key data.
TYPE_EC_FP_PRIVATE_TRANSIENT_DESELECT	3.0.1	Key object which implements the interface type ECPrivateKey for EC operations over large prime fields, with CLEAR_ON_DESELECT transient key data.
TYPE_EC_FP_PARAMETERS	3.1	Key object containing domain parameters which implements interface types ECKey and Key
TYPE_KOREAN_SEED_TRANSIENT_RESET	2.2.2	Key object which implements interface type KoreanSEEDKey with CLEAR_ON_RESET transient key data.
TYPE_KOREAN_SEED_TRANSIENT_DESELECT	2.2.2	Key object which implements interface type KoreanSEEDKey with CLEAR_ON_DESELECT transient key data.
TYPE_KOREAN_SEED LENGTH_KOREAN_SEED_128	2.2.2	Key object which implements interface type KoreanSEEDKey with persistent key data. Korean Seed Key Length = 128.
TYPE_HMAC_TRANSIENT_RESET	2.2.2	Key object which implements interface type HMACKey with CLEAR_ON_RESET transient key data.
TYPE_HMAC_TRANSIENT_DESELECT	2.2.2	Key object which implements interface type HMACKey with CLEAR_ON_DESELECT transient key data.
TYPE_HMAC LENGTH_HMAC_SHA_1_BLOCK_64	2.2.2	Key object which implements interface type HMACKey with persistent key data. HMAC Key Length = 64.
TYPE_HMAC LENGTH_HMAC_SHA_256_BLOCK_64	2.2.2	Key object which implements interface type HMACKey with persistent key data. HMAC Key Length = 64.
TYPE_HMAC LENGTH_HMAC_SHA_384_BLOCK_64	2.2.2	Key object which implements interface type HMACKey with persistent key data. HMAC Key Length = 64.
TYPE_HMAC LENGTH_HMAC_SHA_512_BLOCK_64	2.2.2	Key object which implements interface type HMACKey with persistent key data. HMAC Key Length = 64.
TYPE_DH_PRIVATE_TRANSIENT_DESELECT	3.0.5	Key object which implements the interface type DHPrivateKey for DH operations, .
TYPE_DH_PRIVATE_TRANSIENT_RESET	3.0.5	Key object which implements the interface type DHPrivateKey for DH operations.

TYPE_DH_PUBLIC_TRANSIENT_DESELECT	3.0.5	Key object which implements the interface type DHPublicKey for DH operations, .
TYPE_DH_PUBLIC_TRANSIENT_RESET	3.0.5	Key object which implements the interface type DHPublicKey for DH operations.
TYPE_DH_PARAMETERS	3.1	Key object containing domain parameters which implements interface types DHKey and Key
TYPE_GENERIC_SECRET	3.1	Key object which implements interface type GenericSecretKey
TYPE_SM4	3.1	Key object which implements interface type SM4Key
TYPE_XEC	3.1	Extended Key object which implements the interface type XECKey
ALG_TYPE_DH_PRIVATE	3.0.5	Algorithmic key type ALG_TYPE_DH_PRIVATE choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method.
ALG_TYPE_DH_PUBLIC	3.0.5	Algorithmic key type ALG_TYPE_DH_PUBLIC choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method.
ALG_TYPE_GENERIC_SECRET	3.1	Algorithmic key type ALG_GENERIC_SECRET choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method.
ALG_TYPE_SM4	3.1	Algorithmic key type ALG_SM4 choice for the algorithmicKeyType parameter of the buildKey(byte, byte, short, boolean) method.
LENGTH_DH_1024	3.0.5	DH Key Length = 1024.
LENGTH_DH_2048	3.0.5	DH Key Length = 2048.
LENGTH_SM4	3.1	SM4 Key Length = 128.
javacard.security.KeyPair ALG_RSA	**introduced in JavaCard version**	**Description**
ALG_RSA LENGTH_RSA_512	2.1.1	KeyPair object containing a RSA key pair. RSA Key Length = 512.
ALG_RSA LENGTH_RSA_736	2.2.0	KeyPair object containing a RSA key pair. RSA Key Length = 736.
ALG_RSA LENGTH_RSA_768	2.1.1	KeyPair object containing a RSA key pair. RSA Key Length = 768.
ALG_RSA LENGTH_RSA_896	2.2.0	KeyPair object containing a RSA key pair. RSA Key Length = 896.
ALG_RSA LENGTH_RSA_1024	2.1.1	KeyPair object containing a RSA key pair. RSA Key Length = 1024.
ALG_RSA LENGTH_RSA_1280	2.2.0	KeyPair object containing a RSA key pair. RSA Key Length = 1280.
ALG_RSA LENGTH_RSA_1536	2.2.0	KeyPair object containing a RSA key pair. RSA Key Length = 1536.

ALG_RSA LENGTH_RSA_1984	2.2.0	KeyPair object containing a RSA key pair. RSA Key Length = 1984.
ALG_RSA LENGTH_RSA_2048	2.1.1	KeyPair object containing a RSA key pair. RSA Key Length = 2048.
ALG_RSA LENGTH_RSA_3072	3.0.5	KeyPair object containing a RSA key pair. RSA Key Length = 3072.
ALG_RSA LENGTH_RSA_4096	3.0.1	KeyPair object containing a RSA key pair. RSA Key Length = 4096.
javacard.security.KeyPair ALG_RSA_CRT	**introduced in JavaCard version**	**Description**
ALG_RSA_CRT LENGTH_RSA_512	2.1.1	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 512.
ALG_RSA_CRT LENGTH_RSA_736	2.2.0	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 736.
ALG_RSA_CRT LENGTH_RSA_768	2.1.1	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 768.
ALG_RSA_CRT LENGTH_RSA_896	2.2.0	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 896.
ALG_RSA_CRT LENGTH_RSA_1024	2.1.1	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 1024.
ALG_RSA_CRT LENGTH_RSA_1280	2.2.0	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 1280.
ALG_RSA_CRT LENGTH_RSA_1536	2.2.0	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 1536.
ALG_RSA_CRT LENGTH_RSA_1984	2.2.0	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 1984.
ALG_RSA_CRT LENGTH_RSA_2048	2.1.1	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 2048.
ALG_RSA_CRT LENGTH_RSA_3072	3.0.5	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 3072.
ALG_RSA_CRT LENGTH_RSA_4096	3.0.1	KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form. RSA Key Length = 4096.
javacard.security.KeyPair ALG_DSA	**introduced in JavaCard version**	**Description**
ALG_DSA LENGTH_DSA_512	2.1.1	KeyPair object containing a DSA key pair. DSA Key Length LENGTH_DSA_512 = 512.

ALG_DSA LENGTH_DSA_768	2.1.1	KeyPair object containing a DSA key pair. DSA Key Length LENGTH_DSA_512 = 768.
ALG_DSA LENGTH_DSA_1024	2.1.1	KeyPair object containing a DSA key pair. DSA Key Length LENGTH_DSA_512 = 1024.
javacard.security.KeyPair ALG_EC_F2M	**introduced in JavaCard version**	**Description**
ALG_EC_F2M LENGTH_EC_F2M_113	2.2.1	KeyPair object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 113.
ALG_EC_F2M LENGTH_EC_F2M_131	2.2.1	KeyPair object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 131.
ALG_EC_F2M LENGTH_EC_F2M_163	2.2.1	KeyPair object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 163.
ALG_EC_F2M LENGTH_EC_F2M_193	2.2.1	KeyPair object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis. EC Key Length = 193.
javacard.security.KeyPair ALG_EC_FP	**introduced in JavaCard version**	**Description**
ALG_EC_FP LENGTH_EC_FP_112	2.2.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 112.
ALG_EC_FP LENGTH_EC_FP_128	2.2.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 128.
ALG_EC_FP LENGTH_EC_FP_160	2.2.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 160.
ALG_EC_FP LENGTH_EC_FP_192	2.2.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 192.
ALG_EC_FP LENGTH_EC_FP_224	3.0.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 224.
ALG_EC_FP LENGTH_EC_FP_256	3.0.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 256.
ALG_EC_FP LENGTH_EC_FP_384	3.0.1	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 384.
ALG_EC_FP LENGTH_EC_FP_521	3.0.4	KeyPair object containing an EC key pair for EC operations over large prime fields. EC Key Length = 521.
Javacard.security.KeyPair.ALG_DH	**introduced in JavaCard version**	**Description**

ALG_DH LENGTH_DH_1024	3.0.5	KeyPair object containing an DH key pair for modular exponentiation based Diffie Hellman KeyAgreement operations. DH Key Length = 1024.
ALG_DH LENGTH_DH_2048	3.0.5	KeyPair object containing an DH key pair for modular exponentiation based Diffie Hellman KeyAgreement operations. DH Key Length = 2048.
javacard.security.KeyAgreement	**introduced in JavaCard version**	**Description**
ALG_EC_SVDP_DH	2.2.1	Deprecated
ALG_EC_SVDP_DHC	2.2.1	Deprecated
ALG_EC_SVDP_DH_KDF	3.0.1	Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363].
ALG_EC_SVDP_DH_PLAIN	3.0.1	Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363].
ALG_EC_SVDP_DHC_KDF	3.0.1	Elliptic curve secret value derivation primitive, Diffie-Hellman version, with cofactor multiplication and compatibility mode, as per [IEEE P1363].
ALG_EC_SVDP_DHC_PLAIN	3.0.1	Elliptic curve secret value derivation primitive, Diffie-Hellman version, with cofactor multiplication and compatibility mode, as per [IEEE P1363].
ALG_DH_PLAIN	3.0.5	Diffie-Hellman (DH) secret value derivation primitive as per NIST Special Publication 800-56Ar2.
ALG_EC_PACE_GM	3.0.5	Elliptic curve Generic Mapping according to TR03110 v2.
ALG_EC_SVDP_DH_PLAIN_XY	3.0.5	Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363].
ALG_SM2	3.1	SM2 Key Exchange protocol, using named curve key NamedParameterSpec.SM2, as defined in GM/T 0003.3-2012
ALG_XDH	3.1	X25519 and X448 Diffie-Hellman key agreement protocol, using named curves keys NamedParameterSpec.X25519 or NamedParameterSpec.X448, as defined in RFC 7748
ALG_SM2_WITH_CONFIRMATION	3.2	SM2 Key Exchange protocol with the optional confirmation values, using named curve key NamedParameterSpec.SM2, as defined in GM/T 0003.3-2012.
javacard.security.Checksum	**introduced in JavaCard version**	**Description**
ALG_ISO3309_CRC16	2.2.1	ISO/IEC 3309 compliant 16 bit CRC algorithm.
ALG_ISO3309_CRC32	2.2.1	ISO/IEC 3309 compliant 32 bit CRC algorithm.

javacard.security. NamedParameterSpec	introduced in JavaCard version	Description
BRAINPOOLP192R1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP192r1 curve as defined in RFC 5639.
BRAINPOOLP192T1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP256t1 curve as defined in RFC 5639.
BRAINPOOLP320R1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP320r1 curve as defined in RFC 5639.
BRAINPOOLP320T1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP320t1 curve as defined in RFC 5639.
BRAINPOOLP384R1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP384r1 curve as defined in RFC 5639.
BRAINPOOLP384T1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP384t1 curve as defined in RFC 5639.
BRAINPOOLP512R1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP512r1 curve as defined in RFC 5639.
BRAINPOOLP512T1	3.1	Parameters for ECDSA and ECDH key agreement operations using brainpoolP512t1 curve as defined in RFC 5639.
ED25519	3.1	Parameters for EdDSA operations using curve25519 as defined in RFC 8032.
ED448	3.1	Parameters for EdDSA operations using curve448 as defined in RFC 8032
FRP256V1	3.1	Parameters for ECDSA and ECDH key agreement operations using FRP256v1 curve parameters as defined by ANSSI
SECP192R1	3.1	Parameters for ECDSA and ECDH key agreement operations using secp192r1 curve as defined in FIPS 186-4.
SECP224R1	3.1	Parameters for ECDSA and ECDH key agreement operations using secp224r1 curve as defined in FIPS 186-4.
SECP256R1	3.1	Parameters for ECDSA and ECDH key agreement operations using secp256r1 curve as defined in FIPS 186-4.
SECP384R1	3.1	Parameters for ECDSA and ECDH key agreement operations using secp384r1 curve as defined in FIPS 186-4.
SECP521R1	3.1	Parameters for ECDSA and ECDH key agreement operations using secp521r1 curve as defined in FIPS 186-4.

SM2	3.1	Parameters for signature, key exchange and encryption operations using SM2 curve as defined in GM/T 0003.5-2012.
X25519	3.1	Parameters for ECDH key agreement operations using curve25519 as defined in RFC 7748.
X448	3.1	Parameters for ECDH key agreement operations using curve448 as defined in RFC 7748.
javacardx.security.derivation.DerivationFunction	**introduced in JavaCard version**	**Description**
ALG_KDF_ANSI_X9_63	3.1	Algorithm implementing the KDF Key Derivation Function defined in the standard ANSI X9.63.
ALG_KDF_COUNTER_MODE	3.1	Algorithm implementing KDF in Counter Mode defined in NIST SP 800-108 (Recommendation for Key Derivation Using Pseudorandom Functions)
ALG_KDF_DPI_MODE	3.1	Algorithm implementing KDF in Double Pipeline Iteration Mode defined in NIST SP 800-108 (Recommendation for Key Derivation Using Pseudorandom Functions)
ALG_KDF_FEEDBACK_MODE	3.1	Algorithm implementing KDF in Feedback Mode defined in NIST SP 800-108 (Recommendation for Key Derivation Using Pseudorandom Functions)
ALG_KDF_HKDF	3.1	Algorithm implementing the HKDF Key Derivation function defined in IETF RFC 5869.
ALG_KDF_ICAO_MRTD	3.1	Algorithm implementing the KDF Key Derivation Function defined in the standard ICAO MRTD Doc 9303.
ALG_KDF_IEEE_1363	3.1	Algorithm implementing the KDF1 Key Derivation Function defined in the standard IEEE 1363-2000.
ALG_PRF_TLS11	3.1	Algorithm implementing the TLS version 1.1 Pseudo Random Function defined in IETF RFC 4346.
ALG_PRF_TLS12	3.1	Algorithm implementing the TLS version 1.2 Pseudo Random Function defined in IETF RFC 5246.
ALG_HKDF_EXPAND_LABEL_TLS13	3.2	Algorithm implementing the HKDF-Expand-Label Key Derivation defined in TLS 1.3 - IETF RFC 8446.

APPENDIX 4: GLOSSARY

Term	Definition
AID	Application identifier, an ISO-7816 data format used for unique identification of Java Card applets (and certain kinds of files in card file systems). The Java Card platform uses the AID data format to identify applets, packages and CAP files. AIDs are administered by the International Opens Organization (ISO), so they can be used as unique identifiers. AIDs are also used in the security policies (see "Context" below): applets' AIDs are related to the selection mechanisms, CAP files' AIDs are used in the enforcement of the firewall. Note: although they serve different purposes, they share the same namespace.
APDU	Application Protocol Data Unit, an ISO 7816-4 defined communication format between the card and the off-card applications. Cards receive requests for service from the CAD in the form of APDUs. These are encapsulated in Java Card System by the javacard.framework.APDU class ([JCAPI22]). APDUs manage both the selection-cycle of the applets (through Java Card RE mediation) and the communication with the Currently selected applet.
APDU buffer	The APDU buffer is the buffer where the messages sent (received) by the card depart from (arrive to). The Java Card RE owns an APDU object (which is a Java Card RE Entry Point and an instance of the javacard.framework.APDU class) that encapsulates APDU messages in an internal byte array, called the APDU buffer. This object is made accessible to the currently selected applet when needed, but any permanent access (out-of selection-scope) is strictly prohibited for security reasons.
Applet	The name given to any Java Card technology-based application. An applet is the basic piece of code that can be selected for execution from outside the card. Each applet on the card is uniquely identified by its AID.
Applet deletion ager	The on-card component that embodies the mechanisms necessary to delete an applet or library and its associated data on smart cards using Java Card technology.

BCV	The bytecode verifier is the software component performing a static analysis of the code to be loaded on the card. It checks several kinds of properties, like the correct format of CAP files and the enforcement of the typing rules associated to bytecodes. If the component is placed outside the card, in a secure environment, then it is called an off-card verifier. If the component is part of the embedded software of the card it is called an on-card verifier.
CAD	Card Acceptance Device or card reader. The device where the card is inserted, and which is used to communicate with the card. Unless explicitly said otherwise, in this document, CAD covers PCD.
CAP file	A file in the Converted applet format. A CAP file contains a binary representation of one or several packages of classes that can be installed on a device and used to execute the packages' classes on a Java Card virtual machine. A CAP file can contain a user library, or the code of one or more applets.
Class	In object-oriented programming languages, a class is a prototype for an object. A class may also be considered as a set of objects that share a common structure and behavior. Each class declares a collection of fields and methods associated to its instances. The contents of the fields determine the internal state of a class instance, and the methods the operations that can be applied to it. Classes are ordered within a class hierarchy. A class declared as a specialization (a subclass) of another class (its super class) inherits all the fields and methods of the latter. Java platform classes should not be confused with the classes of the functional requirements (FIA) defined in the CC.
Context	A context is an object-space partition associated to a CAP file. Applets in Java technology-based packages contained within the same CAP file belong to the same context. The firewall is the boundary between contexts (see "Current context").
Current context	The Java Card RE keeps track of the current Java Card System context (also called "the active context"). When a virtual method is invoked on an object, and a context switch is required and permitted, the current context is changed to correspond to the context of the applet that owns the object. When that method returns, the previous context is restored. Invocations

	of static methods have no effect on the current context. The current context and sharing status of an object together determine if access to an object is permissible.
Currently selected applet	The applet has been selected for execution in the current session. The Java Card RE keeps track of the currently selected Java Card applet. Upon receiving a SELECT command from the CAD or PCD with this applet's AID, the Java Card RE makes this applet the currently selected applet over the I/O interface that received the command. The Java Card RE sends all further APDU commands received over each interface to the currently selected applet on this interface ([JCRE3], Glossary).
Default applet	The applet that is selected after a card reset or upon completion of the PICC activation sequence on the contactless interface ([JCRE3], §4.2).
DPA	Differential Power Analysis is a form of side channel attack in which an attacker studies the power consumption of a cryptographic hardware device such as a smart card.
Embedded Software	Pre-issuance loaded software.
Firewall	The mechanism in the Java Card technology for ensuring applet isolation and object sharing. The firewall prevents an applet in one context from unauthorized access to objects owned by the Java Card RE or by an applet in another context.
Installer	The installer is the on-card application responsible for the installation of applets on the card. It may perform (or delegate) mandatory security checks according to the card issuer policy (for bytecode-verification, for instance), loads and link CAP files 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 ([JCRE3],§11.1.6).
Interface	A special kind of Java programming language class, which declares methods, but provides no implementation for them. A class may be declared as being the implementation of an interface, and in this case must contain an implementation for each of the

	methods declared by the interface (See also shareable interface).
Java Card RE	The runtime environment under which Java programs in a smart card are executed. It is in charge of all the management features such as applet lifetime, applet isolation, object sharing, applet loading, applet initializing, transient objects, the transaction mechanism and so on.
Java Card RE Entry Point	An object owned by the Java Card RE context but accessible by any application. These methods are the gateways through which applets request privileged Java Card RE services: the instance methods associated to those objects may be invoked from any context, and when that occurs, a context switch to the Java Card RE context is performed. There are two categories of Java Card RE Entry Point Objects: Temporary ones and Permanent ones. As part of the firewall functionality, the Java Card RE detects and restricts attempts to store references to these objects.
Java Card RMI	Java Card Remote Method Invocation is the Java Card System version 2.2 and 3 Classic Edition mechanism enabling a client application running on the CAD platform to invoke a method on a remote object on the card.
Java Card System	Java Card System includes the Java Card RE, the Java Card VM, the Java Card API and the installer.
Java Card VM	The embedded interpreter of bytecodes. The Java Card VM is the component that enforces separation between applications (firewall) and enables secure data sharing.
Logical channel	A logical link to an application on the card. A new feature of the Java Card System, version 2.2 and 3 Classic Edition, that enables the opening of simultaneous sessions with the card, one per logical channel. Commands issued to a specific logical channel are forwarded to the active applet on that logical channel. Java Card platform, version 2.2.2 and 3 Classic Edition, enables opening up to twenty logical channels over each I/O interface (contacted or contactless). Logical channels are a Java Card specification version 2.2-3.1 feature, and optional for version 3.2.
NVRAM	Non-Volatile Random Access Memory, a type of memory that retains its contents when power is turned off.

Object deletion	The Java Card System version 2.2 and 3 Classic Edition mechanism ensures that any unreferenced persistent (transient) object owned by the current context is deleted. The associated memory space is recovered for reuse prior to the next card reset.
Package	A package is a namespace within the Java programming language that may contain classes and interfaces. A package defines either a user library, or one or more applet definitions. A package is divided in two sets of files: export files (which exclusively contain the public interface information for an entire package of classes, for external linking purposes; export files are not used directly in a Java Card virtual machine) and CAP files where CAP file may contain one or more packages.
PCD	Proximity Coupling Device. The PCD is a contactless card reader device.
PICC	Proximity Card. The PICC is a card with contactless capabilities.
RAM	Random Access Memory, is a type of computer memory that can be accessed randomly
SCP	Smart Card Platform.It is comprised of the integrated circuit, the operating system and the dedicated software of the smart card.
Shareable interface	An interface declaring a collection of methods that an applet accepts to share with other applets. These interface methods can be invoked from an applet in a context different from the context of the object implementing the methods, thus "traversing" the firewall.
SIO	An object of a class implementing a shareable interface.
Subject	An active entity within the TOE that causes information to flow among objects or change the system's status. It usually acts on the behalf of a user. Objects can be active and thus are also subjects of the TOE.
SWP	The Single Wire Protocol is a specification for a single-wire connection between the SIM card and a Near Field Communication (NFC) chip in a mobile handset
Transient object	An object whose contents are not preserved across CAD sessions. The contents of these objects are cleared at the end of the current CAD session or when

	a card reset is performed. Writes to the fields of a transient object are not affected by transactions.
User	Any application interpretable by the Java Card RE. That also covers the packages. The associated subject(s), if applicable, is (are) an object(s) belonging to the javacard.framework.applet class.

Index

End of Document