Trusted Logic

JCLX80jTOP20ID/JCLX36jTOP20ID Security Target Lite

Emission Date	:	December 17th, 2008
Document Type	:	Technical report
Ref./Version	:	CP-2007-RT-075-1.4
Number of pages	:	122 (including two cover pages)

LEGAL NOTICE

COPYRIGHT NOTICE

DISCLAIMER OF WARRANTY

This Document is provided "as is" and all express or implied conditions, representations and warranties, including, but not limited to, any implied warranty of merchantability, fitness for a particular purpose or non-infringement, are disclaimed, except to the extent that such disclaimers are held to be legally invalid. Trusted Logic shall not be liable for any special, incidental, indirect or consequential damages of any kind, arising out of or in connection with the use of this Document.

Table of contents

TABLE OF CONTENTS 3

TABLE OF FIGURES 5

1 INTRODUCTION 6

1.1. IDENTIFICATION OF THIS DOCUMENT 6
1.2. IDENTIFICATION OF THE TOE..6
1.3. DIFFUSION LIST 6
1.4. REVISIONS AND COMMENTS 6
1.5. CC CONFORMANCE..7
1.6. PP CLAIMS 7

2 OVERVIEW 8

2.1. TYPOGRAPHIC CONVENTIONS 8
2.2. ASSOCIATED DOCUMENTS 9
2.3. ACRONYMS 11
2.4. GLOSSARY 12

3 TOE DESCRIPTION 17

3.1. THE TARGET OF EVALUATION 17
3.1.1. Bytecode Verification..19
3.1.2. Installation of Executable Files 19
3.1.3. Installation of Application instances 20
3.1.4. Deletion of Application instances and Executable Files...21
3.1.5. Card Management..21
3.1.6. Services to the Applets 22
3.1.7. Smart Card Platform: Operating System, Dedicated Software and Chip 22
3.2. LIMITS OF THE TOE 22
3.2.1. Scope of Evaluation..22
3.2.2. Users and Roles..23
3.2.3. The TOE in the Smart Card's Life Cycle..25
3.3. TOE INTENDED USAGE..27

4 TOE SECURITY ENVIRONMENT 28

4.1. ASSETS..28
4.1.1. Card Management Assets..28
4.1.2. Runtime Environment Assets...30
4.2. SUBJECTS...31
4.2.1. IT Entities..32
4.3. ASSUMPTIONS...33
4.3.1. Assumptions on the Embedded Software 33
4.3.2. Assumptions on the IC 34
4.4. THREATS..35
4.4.1. Card Management..35
4.4.2. Runtime Environment..39
4.4.3. Integrated Circuit Threats 43
4.5. ORGANISATIONAL SECURITY POLICIES 45
4.5.1. Embedded Software OSP 45
4.5.2. Integrated Circuit OSP..47

5 SECURITY OBJECTIVES... 48

5.1. SECURITY OBJECTIVES FOR THE TOE 48
5.1.1. Objectives for the Card Manager...48
5.1.2. Objectives for the Runtime Environment..52

5.1.3. Objectives for the Integrated Circuit......55
5.2. SECURITY OBJECTIVES FOR THE ENVIRONMENT......57

6 IT SECURITY REQUIREMENTS 60

6.1. TOE SECURITY FUNCTIONAL REQUIREMENTS 60
6.1.1. Card Management......60
6.1.2. Runtime Environment......82
6.1.3. Smart Card Platform......100
6.2. TOE SECURITY ASSURANCE REQUIREMENTS 107
6.3. SECURITY REQUIREMENTS FOR THE IT ENVIRONMENT 107
6.3.1. IT environment functional requirements 107
6.4. SECURITY REQUIREMENTS FOR THE NON-IT ENVIRONMENT 110
6.4.1. Non-IT environment functional requirements 110

7 TOE SUMMARY SPECIFICATION 111

7.1. TOE SECURITY FUNCTIONS 111
7.1.1. Card Management...... 111
7.1.2. Runtime Environment...... 112
7.1.3. Integrated Circuit TSFs...... 116

8 EXTENDED REQUIREMENTS...... 118

8.1. EXTENDED FAMILIES 118
8.1.1. Extended family FCS_RND - Generation of random numbers...... 118
8.2. EXTENDED COMPONENTS...... 118
8.2.1. Extended component FCS_RND.1 118
8.2.2. Extended component FPT_TST.2 119

INDEX 120

Table of figures

Figure 1: The TOE and its environment... 18
Figure 2: Development and operation lifecycle.. 26

1 Introduction

This chapter identifies this Security Target, its TOE, presents its general structure, and introduces the references, notation conventions, and technical terms to be used in the following chapters.

1.1. Identification of this document

Author	Trusted Logic SA
Address	5, rue du Bailliage 78000 Versailles - France
Title	JCLX80jTOP20ID/JCLX36jTOP20ID Security Target Lite
Version	1.4
Keywords	Smart Card; Java Card; GlobalPlatform

1.2. Identification of the TOE

Commercial names	JCLX80jTOP20ID/JCLX36jTOP20ID
TOE version	IFXv#27_0.1 with software revision v1.4
IC identifiers	SLE66CLX800PE-m1581-e13/a14 and SLE66CLX360PE-m1587-e13/a14

1.3. Diffusion List

No restrictions (public document).

1.4. Revisions and Comments

Version	Issue date	Comments
1.0	21/08/2006	Initial version.
1.1	31/03/2008	Preliminary version, after evaluation of the full security target.
1.2	08/12/2008	Version submitted to SERMA and DCSSI.
1.3	13/12/2008	Version after first SERMA´s feedback.
1.4	17/12/2008	Certified version.

1.5. CC Conformance

This Security Target claims conformance to the following documents defining the ISO/IEC 15408:2005 standard:

- Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, August 2005, Version 2.3, CCMB-2005-08-001.
- Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements, August 2005, Version 2.3, CCMB-2005-08-002.
- Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance requirements, August 2005, Version 2.3, CCMB-2005-08-003.
- Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, August 2005, Version 2.3, CCMB-2005-08-004.

Conformance to ISO/IEC 15408:2005 is claimed as follows:

- Part 1: conformant
- Part 2: extended with the FCS_RND.1 and the FPT_TST.2 families. All the other security requirements have been drawn from the catalogue of requirements in Part 2 of ISO/IEC 15408:2005.
- Part 3: EAL5 augmented with ALC_DVS.2 and AVA_VLA.4 with SOF-high as the minimum strength for the security functions.

1.6. PP Claims

This security target has been inspired from [JCSPP]. However, as Remote Method Invocation and the use of several logical channels are not included in the evaluation scope, the conformance to that PP is not claimed.

2 Overview

This document is the Security Target of the Java Trusted Open Platform (jTOP). By open smart card is meant a smart card enabling the possibility of enlarging and restricting the set of applications installed on the card, either in the pre-issuance or post-issuances phases of its life-cycle. The platform is compliant with Java Card 2.2.1 and VISA GlobalPlatform 2.1.1–Configuration 2 standards.

One of the main security goals of jTOP is to counter the unauthorized disclosure or modification of the code and data of the application instances installed on the card, including the application's code, keys and PINs. In order to achieve these goals, jTOP provides the following key security features:

- Logical separation of the data used by different applications (Java Card firewall)
- Runtime monitoring of applet execution (Defensive Virtual Machine)
- Verification of the origin and integrity of Executable Load Files prior to installation.

In addition to this, jTOP provides basic security services to the applications such as:

- Management of cryptographic keys and PINs
- Symmetric and asymmetric cryptography
- Atomic transactions
- Secure communication channels with the terminal

This document has been conceived to prepare a Common Criteria evaluation using the "compositional approach" described in [ETR] and detailed in [ETRSC] for the smart cards. This approach consists in starting from an integrated circuit that has been independently certified by the Chip Manufacturer, and performing an evaluation of the product resulting from embedding a piece of software into it, which make use of some of the results issued from the evaluation of the chip. The integrated circuit has been evaluated according to the [SSVG] Protection Profile.

2.1. Typographic Conventions

A «T», like in T.INSTALL, prefixes the name of the threats. Similarly, an "O" prefixes the security objectives, the string «OSP» prefixes the organizational security policies, the letter "A" prefixes the assumptions and the letter "D" prefixes the assets. The instances of the security functional requirements in [CC2] are identified by the name of the instantiated component, followed by a suffix, like in FDP_ACC.1-FIREWALL.

This security target also contains threats, assumptions, organizational security policies, security objectives, security functional requirements and TOE security functions of the chip security target [ICST]. Only those items that directly or indirectly relate to the security issues addressed in [JCSPP] have been selected. The suffix "IC" is used for the security functional requirements coming from that Security Target.

2.2. Associated Documents

[CC1] *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model*, Version 2.3, August 2005, CCMB-2005-08-001.

[CC2] *Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements*, Version 2.3, August 2005. CCMB-2005-08-002.

[CC3] *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance requirements*, Version 2.3, August 2005. CCMB-2005-08-003.

[CEM] *Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology*, Version 2.3. August 2005. CEM-2005-08-004.

[DCSSI2791] *Mécanismes cryptographiques – Règles et recommandations concernant le choix et le dimensionement des mécanismes cryptographiques de niveau de robustesse « standard »*. DCSSI, version 1.02, November 19th 2004.

[ETR] *ETR-lite for composition*, Version 1.1, July 2002. Available at the address www.commoncriteriaportal.org.

[ETRSC] *ETR-lite for composition, Annex A, Composite Smart Card Evaluation: Recommended Best Practice*, Version 1.2, March 2002. Available at the address www.commoncriteriaportal.org.

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

[GPCS] *GlobalPlatform 2.1.1 Card Specification (March 2003), including Amendment A and Errata Precision List 1.3 (december 2004).*

[ICST] *SLE66CLX800PE-m1581-e13/a14 & SLE66CLX360PE-m1587-e13/a14 Security Target with optional libraries RSA2048 V1.5 and ECC v1.1, Version 1.2, January 9th, 2008..*

[JCAPI] *Java Card 2.2.1 Application Programming Interface, Sun Microsystems , October 2003.*

[JCRE] *Java Card 2.2.1 Runtime Environment Specification, Sun Microsystems,, October 2003.*

[JCSPP] *Java Card System Standard 2.2.1 Configuration Protection Profile*, Version 1.0b, August 2003, registered and certified by the French Certification Body (DCSSI) under the reference PP/0305.

[JCSPPD] *Java Card System Defensive Configuration Protection Profile*, Version 1.0b, August 2003, registered and certified by the French Certification Body (DCSSI) under the reference PP/0306.

[JCVM] *Java Card 2.2.1 Virtual Machine Specification, Sun Microsystems, October 2003.*

[MRTD] *PKI for Machine Readable Travel Documents offering ICC Read-Only Access*, International Civil Aviation Organization (ICAO). Version 1.1, October 1st 2004.

[PROFILE] *jTOP Platform -- Profile, Trusted Logic, CP-2007-RT-246-27/1.1/SERMA*

[SSCD] *Application Interface for smart cards used as Secure Signature Creation Devices*, European Committee for Standardization (CEN), CWA 14890-1:2004 (E), 22snd December 2003.

[SSVG] *Smartcard IC Platform Protection Profile*, Version 1.0, July 2001, registered at the BSI under the reference BSI-PP-0002.

[TR03110] Advanced Security Mechanisms for Machine Readable Travel Documents – Extended Access Control, BSI-TR-03110.

[VCPG] *VISA GlobalPlatform 2.1.1 Card Production Guide, Version 1.01, March 2005*.

[VGP] *Visa GlobalPlatform 2.1.1 Card Implementation Requirements, Version 2.0, July 2007.*

2.3. Acronyms

The following acronyms are used in this document:

Acronym	Meaning
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer To Reset
CAD	Card Acceptance Device
CC	Common Criteria
CCM	Card Content Management
CLA	Instruction class (of an APDU command)
CPLC	Card Production Life Cycle Data
CVM	Cardholder Verification Method
DAP	Data Authentication Pattern
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DPA	Differential Power Analysis
EEPROM	Electrically Erasable Programmable Read Only Memory
EMA	Electro-Magnetic Analysis
EPA	Emanation Power Analysis
GP	GlobalPlatform
INS	Instruction code (of an APDU command)
ISD	Issuer Security Domain
JAR	Java Archive file
JCSPP	Java Card System Protection Profile
JCAPI	Java Card Application Programming Interface
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
jTOP	Java Trusted Open Platform
MAC	Message Authentication Code
OPEN	Open Platform Environment
OS	Operating System
PIN	Personal Identification Number
PP	Protection Profile
ROM	Read Only Memory
RSA	Rivest Shamir Adleman
RTE	Run Time Environment
SCP	Smart Card Platform
SCP01	Secure Channel Protocol 01
SCP02	Secure Channel Protocol 02
SAR	Security Assurance Requirement
SF	Security Function
SFR	Security Functional Requirement
SPA	Simple Power Analysis
ST	Security Target
TOE	Target of Evaluation
TSF	TOE Security Functions
VGP	VISA GlobalPlatform

2.4. Glossary

Term	Definition
Applet	An application written in Java Card.
Application instance	Instance of an Executable Module after it has been installed and made selectable.
Application Code Verification	A static analysis of an Executable Module to determine whether it respects the CAP format and satisfies some essential security properties, such as the absence of pointer arithmetic, uncontrolled control jumps, data-structure overflows, etc..
Application Protocol Data Unit (APDU)	Standard communication messaging protocol between a card accepting device and a smart card. See ISO-7816-4.
Application Provider	The institution that owns an Application and is responsible for its behavior.
Application Session	The link between the Application and the external world during a Card Session starting with the Application selection and ending with Application de-selection or termination of the Card Session.
Asymmetric Cryptography	A cryptographic technique that uses two related transformations, a public transformation (defined by the Public Key component) and a private transformation (defined by the Private Key component); these two key components have a property so that it is computationally infeasible to discover the Private Key, even if the Public Key is known.
Card Administrator	The person or organization that has ultimate control of the card, within the policy constraints set by the Card Issuer, with regard to card content and card Life Cycle management.
Card Content	Code and Application information (but not Application data) contained in the card that is under the responsibility of the OPEN e.g. Executable Load Files, Application instances, etc.
Card Enabler	The organization responsible for moving a manufactured TOE to the operational state.
Card Image Number (CIN)	An identifier for a specific smart card.

Cardholder	The end user of the smart card.
Card Issuer	The organization that owns the card and is ultimately responsible for its behavior
Card Manager	Generic term for the card management entities of a GlobalPlatform card i.e. the Open Platform Environment, the Issuer Security Domain.
Card Manufacturer	The organization responsible for integrating the IC containing the embedded software into its carrier, in accordance with the Card Issuer's requirements, to produce a complete card ready for delivery to the Card Enabler.
Card Session	The period of time during which the card receives power supply from the terminal without receiving a card reset signal.
Card Unique Data	Data that uniquely identifies a card, made of the Card Image Number and a code identifying the Card Issuer.
Card Production Life Cycle Data	A record that uniquely identifies the smart card and the actors involved in its manufacturing and personalization.
Cardholder Verification Method (CVM)	A method to ensure that the person presenting the card is the person to whom the card was issued.
Chip Manufacturer	The organization responsible for embedding the software of the OS, RTE and GP in the IC ("masking process").
Closed Mode	A mode in which the card restricts card content management operations. When the card is in the Closed Mode it rejects loading more Executable Load Files. There are two possible closed modes: Java Card Static and Native Card.
Embedded Software	The piece of executable code that is masked on the ROM and written in the EEPROM memories of the integrated circuit. It comprises the Operating System, the Runtime Environment, the Card Manager and the bytecode of the installed Java Card Packages.
Executable File	Actual on-card container of one or more Executable Modules. It may reside in immutable persistent memory or may be created in mutable persistent memory as the resulting image of an Executable Load File.
Executable Load File	An Executable File that is in transit to the smart card.
Executable Module	The on-card executable code of a single Application present within an Executable Load File.

Export File	A binary representation of the type and access modifiers of an Executable File in the CAP format. If B is a CAP file that imports methods or fields of a CAP file A, then the Export File of *A* contains all the information required to perform the bytecode verification of *B*.
GloabalPlatform Registry	A container of information related to Card Content management.
Host	The back end system that supports the smart card. Hosts perform functions such as authorization and authentication, card administration, download of post-issuance Application code and data and transactional processing
Initialization Data	Any data supplied by the Platform Developer that is injected into the non-volatile memory of the IC by the IC Manufacturer. These data are for instance used for initializing the platform, and to enforce traceability and secure shipment between phases.
Issuer Security Domain	On-card entity providing support for the control, security, and communication requirements of the Card Issuer
Java Card Platform	A collective name for all the components of the Embedded Software (OS, RTE and GP) that transform the IC into a Java Card enabled smart card.
Java Card Static	A closed mode in which no more Executable Load Files may be loaded on the card.
Java Card System	The term used in [JCSPP] to refer to the Runtime Environment, plus those parts of the Card Manager corresponding to the Installer and the Applet Deletion Manager.
Masking Process	The process of embedding the binary code of the Operating System, the Runtime Environment, the Card Manager and a collection of applets into the IC chip.
Message Authentication Code (MAC)	A symmetric cryptographic transformation of data that provides data origin authentication and data integrity.
Mutable Persistent Memory	Memory that can be modified.
Native Card Mode	A closed mode in which the card behaves as a native card. GlobalPlatform commands are rejected when the card is in this mode.
Object	An entity on which a Security Policy is enforced.
Open Platform Environment (OPEN)	The on-card piece of software that manages the GlobalPlatform Registry.
Post-Issuance	Phase following the card being issued to the Cardholder.

Platform Developer	The organization responsible for developing the code of the basic OS, RTE and GP software.
Platform Personalization Data	Any data supplied relative to the Card Issuer that is injected into the non-volatile memory of the smart card by the Card Enabler. These data are for instance used to personalize the platform with the Card Issuer’s keys, for traceability purposes, and to secure shipment between phases.
Pre-Issuance	Phase prior to the card being issued to the Cardholder.
Private Key	The private component of an asymmetric key pair.
Public key	The public component of an asymmetric key pair.
Retry Limit	The maximum number of times an invalid CVM value can be presented prior to the CVM handler prohibiting further attempts to present a CVM value.
Retry Counter	A counter, used in conjunction with the Retry Limit, to determine when attempts to present a CVM value shall be prohibited.
Secret key	A private key. In GlobalPlatform specification, this term refers to a key used to generate a Session Keys during the initiation of a Secure Channel.
Session key	A key whose lifetime is a card session. In GlobalPlatform specifications, this term refers to the key associated to a Secure Channel and which is used for a secure communication session.
Secure Channel	A communication mechanism between an off-card entity and a card that provides a level of assurance, to one or both entities.
Secure Channel Session	A session, during an Application Session, starting with the Secure Channel Initiation and ending with a Secure Channel Termination or termination of either the Application Session or Card Session.
Security Attribute	A logical entity used by a Security Policy to determine whether the outcome of a requested operation may succeed.
Security Domain	On-card entity providing support for the control, security, and communication requirements of the Application Provider.
Security Policy	A set of rules that regulate how certain assets are managed, protected and/or distributed.
Subject	The entity within the Platform (e.g. Issuer Security Domain, RTE) that acts on behalf of a User to perform some operation on an Object within the scope of a Security Policy.
Symmetric Cryptography	A cryptographic technique that uses the same secret key for both the originator's and the recipient's transformation.

User	Either an Application (via GP API or JC API) or an off-card entity (via an APDU command) that makes a request to a Subject to perform some operation on an Object within the scope of a Security Policy.
Verification Authority	The organization in charge of verifying that the Java Card applets to be installed on the smart card respect the Platform Developer recommendations and the Card Issuer security policies.

3 TOE Description

This part of the document describes the TOE as an aid to the understanding of its security requirements. It addresses the product type and the general IT features of the TOE.

3.1. The Target of Evaluation

The TOE is a smart card composed of a piece of software embedded into a SLE66CLX800PE/SLE66CLX360PE chip which transforms the card into a secure open platform device for hosting Java Card applications. The different operations involved in the management of those applications are performed in accordance with VISA GlobalPlatform 2.1.1 specifications, Configuration 2. Management operations include the downloading, installation, removal, and selection for execution of Java Card applications, life-cycle management of both the card and the application, and sharing of a global common PIN among all the applications installed on the card.

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 [JCVM]. The Java Card platform is a smart card platform enabled with Java Card technology (also called, for short, a "Java card"). This technology allows for multiple applications to run on a single card and provides facilities for secure interoperability of applications. Applications running on the Java Card platform ("Java Card applications") are called applets.

The TOE is compliant with the version of the Java Card platform specified in [JCVM], [JCRE] and [JCAPI]. It includes the Java Card Virtual Machine (JCVM), the Java Card Runtime Environment (JCRE) and the Java Card Application Programming Interface (JCAPI). As the terminology is sometimes confusing, the term "Java Card System" (JCS) has been introduced in [JCSPP] to designate the set made of the JCRE, the JCVM and the JCAPI. The JCS provides an intermediate layer between the operating system of the card and the applications. This layer allows applications written for one smart card platform enabled with Java Card technology to run on any other such platform.

The JCVM is a bytecode interpreter embedded in the smart card. The JCRE is responsible for card resource management, communication, applet execution, and on-card system and applet security. The JCAPI provides classes and interfaces for the core functionality of a Java Card application. It defines the calling conventions by which an applet may access the JCRE and native services such as, among others, I/O management functions, PIN and cryptographic specific management and the exceptions mechanism. The JCAPI is compatible with formal international standards, such as ISO7816, and industry specific standards, such as EMV (Europay/Master Card/Visa).

The TOE can be configured so that applets can be downloaded and installed on it, even after the smart card has been issued to the Cardholder. This allows Card Issuers to dynamically respond to their customers changing needs. For example, if the Card Issuer decides to upgrade some of the applications offered to the customer, he can make this change without issuing a new card. Moreover, applications from different vendors can coexist in a single card, and they can even share information between them. A smart card application, however, is usually intended to store highly sensitive information, so the sharing of that information must be carefully limited. Applet isolation is achieved through the Java Card Firewall mechanism defined in [JCRE]. 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 the applet that owns it provides a specific interface for that purpose. This access control policy is enforced at runtime by the JCVM. However, applet isolation cannot entirely be granted by the firewall mechanism if certain well-formedness conditions are not satisfied by the applications loaded on the card. A bytecode verifier can statically verify those conditions.

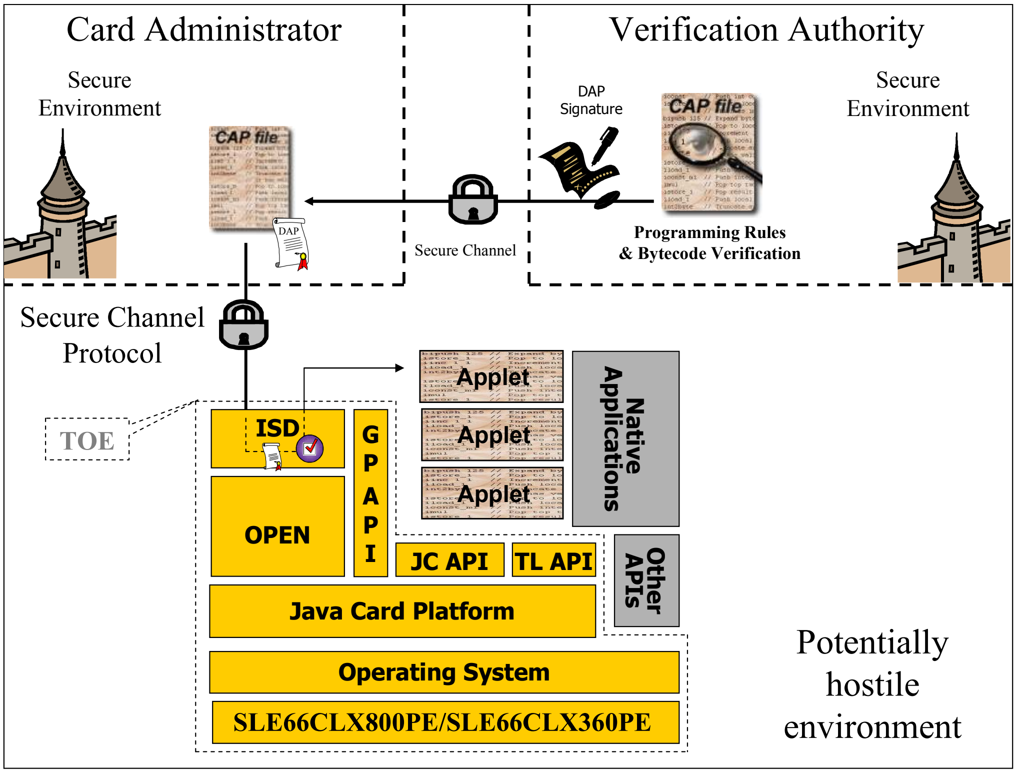

Figure 1: The TOE and its environment

Figure 1 places the different components of the TOE in their environment. Applet source code development is carried on in a Java programming environment. The compilation of that code produces the corresponding class file. This latter file is then processed by the *converter*, which, on the one hand, validates the code, and on the other hand, generates a Converted APplet (CAP) file, the equivalent of a JAR file for the Java Card platform. A CAP file contains an executable binary representation of the classes of a package. A package is a name space within the Java programming language that may contain classes and interfaces. In the context of Java Card technology, it defines either a user library, or one or several applets. This development of the Java source file and its conversion to the CAP format is not relevant for the security of the TOE, and are not included in the figure.

In order to download a new package on the smart card, its code has to be first approved by the Verification Authority. This Verification Authority is responsible for checking that the Applet Developer has enforced all the security recommendations that the Platform Developer has stated in the TOE's User Guide **¡Error! No se encuentra el origen de la referencia.**, and in particular that the CAP format of the package successfully passes a *bytecode verifier* program. Such verifications are performed in a secure physical environment that prevents unauthorized people to access to the applet code. If they are successful, the Verification

Authority electronically signs the CAP file, for instance, using GlobalPlatform's Data Authentication Pattern mechanism (DAP). This signature attests that the Verification Authority has validated the CAP file, and prevents any further modification on it. The Verification Authority then transmits the signed CAP file to the representative of the Card Issuer in charge of loading new applets on the card, called hereto the Card Administrator.

Upon reception of the CAP file, the Card Administrator stores it in its secure environment until the file is downloaded into the card. The Card Administrator transmits the CAP file from its secure environment to the card using a GlobalPlatform's secure channel protocol SCP02. This protocol ensures that the file actually comes from a representative of the Card Issuer and that its integrity has been preserved during the transmission step. The file is received by the Issuer Security Domain (ISD), an on-card representative of the Card Issuer in charge of card management. The Chip Manufacturer may configure the card so that the ISD systematically verifies the DAP signature of the Verification Authority upon reception of a new CAP file[1]. If the signature is correct, the package is *linked*, which makes it possible in turn to install instances of any of the applets defined in the file. During the installation process the applet is registered on the card by using an Application IDentifier (AID). This AID allows unique identification of the applet instance within the card. In particular, the AID is used for selecting the applet instance for execution. The bytecode interpreter residing on the card, usually called the Java Card Virtual Machine, performs the execution of the applet's code.

The following sections further describe the components involved in the use of a Java Card platform. Although some of these components are not part of the TOE, a better understanding of the role they play will in turn allow the reader to grasp the importance of the assumptions that will appear concerning its environment.

3.1.1. Bytecode Verification

The bytecode verifier is a program that performs a static analysis of the bytecode contained in a CAP file. The actual verifications that the bytecode verifier performs are implementation-dependent, it shall at least enforce all the "must clauses" imposed in [JCVM] on the Java Card bytecodes as well as the correctness conditions of the CAP format described in that document.

Bytecode verification is one of the cornerstones of the security architecture of a Java Card platform. It ensures that the bytecodes contained in the CAP file hold up to their intended use. For instance, it ensures that they do not use or forge fake references to memory blocks, perform illegal control jumps, or use return addresses as if they were integers. In particular, the correct working of the Java Card Firewall depends on some of the properties checked during bytecode verification.

Bytecode verification is performed off-card in the secure environment of the Verification Authority, and prior to downloading the CAP file on the card. The DAP signature attests that the CAP file has successfully passed bytecode verification.

3.1.2. Installation of Executable Files

Following Java Card specifications, the [JCSPP] introduces the notion of *Applet Installer* as the application of the platform in charge of downloading, linking and installing new CAP files.

[1] If the card is not configured to verify DAP signatures, it is assumed that there is a secure channel linking the Verification Authority and the Card Administrator that ensures the origin and the integrity of the received CAP file. The description of such secure channel falls beyond the scope of this security target.

In a platform compliant with VISA GlobalPlatform 2.1.1 specifications the so-called Issuer Security Domain (ISD) plays the role of the installer, and CAP files are called Executable Files[2].

When selected, the ISD can receive an Executable Load File to be installed on the card. The file is usually sent along several consecutive APDUs. The ISD collects all the APDUs, links the whole file to the libraries already installed on the card and initializes the static data, if any.

3.1.2.1. Loading

Loading an Executable File into the card includes two main steps: there is first an authentication step by which the Card Administrator and the Issuer Security Domain recognize each other using the SCP02 cryptographic protocol defined by GlobalPlatform. Once the identification step is accomplished, the Executable File is transmitted to the card through some medium that is not supposed to be secure. Due to resource limitations, usually the file is split by the Card Administrator into a list of Application Protocol Data Units (APDUs), which are in turn sent to the card. The Issuer Security Domain controls that it has received all the Executable File pieces in the correct order and without modification. If the card has been configured to do so, it also verifies the DAP signature that the Verification Authority attached to it. In this case, further steps are performed only if the DAP signature is correct.

Loading Executable Files requires the TOE to be configured to support this feature. This feature can be disabled so that the card becomes a *static Java Card Platform*. In this configuration, the platform rejects any attempt of downloading new Executable Files. The set of available applets is the one that can be created from the Java Card packages that have been masked in ROM with the code of the platform and those that have been loaded before moving to the static mode. This operation cannot be undone: once the card becomes static, it cannot rollback to the open configuration again.

3.1.2.2. Linking

The linking process consists of rearranging 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 by direct ones. This is what is elsewhere called the resolution step. In the next step, called the preparation step, the static fields and the statically initialized arrays defined in the CAP file are allocated and initialized. After this step, the CAP file is ready to be executed by the embedded Java Card Virtual Machine.

3.1.3. Installation of Application instances

Each Executable File in the CAP format may contain several Executable Modules, or Applet classes in the Java Card jargon. The actual installation of Applet instances is an independent process that may be delayed in time. Applet installation is then usually separated from the process of loading and linking an Executable File.

GlobalPlatform defines a specific APDU command for creating an instance of one of those Applet classes. This command specifies the Applet Identifier (AID) to be used for selecting the new Application instance and the privileges to be granted to it. The application in charge of processing installation commands is also the Issuer Security Domain.

[2] The term *Executable File* will be preferred in the sequel to its synonyms *Package* and *CAP file*.

3.1.4. Deletion of Application instances and Executable Files

Executable Files and application instances installed on the card may be deleted on demand of the Card Issuer. Three possible deletion cases are considered in Java Card specifications:

- deletion of an application instance, which is the removal of the applet instance and the Java Card objects created by that instance;
- deletion of a Java Card library, which entails the removal of all the card resident components of the Executable File, including code and any associated management structures;
- deletion of an Executable File declaring applets, which is the removal of the card resident code and JCRE structures associated with the Executable File, as well as of all the instances of the applets that the Executable File declares.

Deletion is only possible when no other Executable File or Application instance depends on the item to be deleted.

In [JCSPP], the *Applet Deletion Manager* is introduced as the on-card component in charge of the mechanisms necessary to delete an applet or a CAP file. According to GlobalPlatform specifications, this role is also embodied by the ISD.

3.1.5. Card Management

The Card Manager is the on-card component responsible for the administration of the smart card. It enforces the security policies of the Card Issuer on the card and provides the following supplementary services:

- Life cycle management of both the whole card and of each of the application instances installed on it.
- Management of logical channels, so that the services of several applet instances may all be active at the same time. Logical channels also enable a given applet instance to be active on different logical channels. The applets that support this latter feature are called *multi-selectable* applets[3].
- Dispatching of the APDU commands to the currently active application instances[4].
- Secure communication channels between the application instances on the card (and specially the ISD) and the Card Administrator or the Application Provider.
- A global Cardholder Verification Module that can be shared by all the application instances.

Those services, which are beyond the scope of the [JCSPP], are also included in the scope of this Security Target.

According to GlobalPlatform specifications, the Card Management functions described above are shared between the OP Environment (OPEN) and the ISD. The services they provide are available either through APDU commands or through GlobalPlatform's Application Programming Interface.

Optionally, the Card Manager may be disabled so that the card behaves as a closed-application native card. In this case, after installing the desired applet instances, the ISD becomes not longer selectable and all the external interfaces available for accessing it are disabled. This operation cannot be undone. Once the card is closed, the only card

[3] This service is explicitly excluded from the evaluation scope of this Security Target; see Section 3.2.1.

[4] In the evaluation scope of this Security Target there can be at most one active applet instance at a time; see Section 3.2.1

management command that is supported is the SELECT command specified in ISO7816. Other APDU commands are directly forwarded to the selected application.

3.1.6. Services to the Applets

In order to enforce an adequate level of security, the platform provides the applets with a collection of frequently used, highly secure services, including:

- Cardholder identification and management of Personal Identification Numbers (PIN);
- Symmetric and asymmetric cryptography services, including encryption and decryption, electronic signature generation and verification, and generation of random data and unique hash values;
- Access to some of the internal runtime data areas of the Java Card Virtual Machine, like determining which is the applet that invokes a given service;
- Controlled sharing of class instances between applet instances;
- Allocation of transient arrays, whose lifetime is either restricted to the card session or to the application sessions with the active applets instance of the same Executable File;
- Management of atomic transactions;
- Abstraction of the low-level communications with the CAD;
- Garbage collection of those class instances and arrays that have become unreachable;
- Remote Method Invocation, enabling a client application running on the CAD to directly invoke a method on a class instance on the card, without any explicit processing of an APDU command[5].

These services are available to the applets through the Java Card Application Programming Interface.

3.1.7. Smart Card Platform: Operating System, Dedicated Software and Chip

In the [JCSPP], the Java Card System lays on a *Smart Card Platform*, which is composed of a micro-controller and an operating system. It provides memory management functions with separate interface to RAM and EEPROM, I/O drivers compliant with ISO standards, a low level transaction mechanism, and secure and highly efficient implementation of cryptographic functions. It also contains dedicated software, which provides interface with the integrated circuit.

In this Security Target, the Smart Card Platform is composed of an SLE66CLX800PE/SLE66CLX360PE chip and an operating system developed on top of it.

3.2. Limits of the TOE

This section specifies the components of the smart card that form the Target of Evaluation, and the phases of its life cycle that fall under the scope of the evaluation.

3.2.1. Scope of Evaluation

The scope of the TOE is the SLE66CLX800PE/SLE66CLX360PE chip and the embedded software that makes it a platform for executing Java Card applications. Unlike [JCSPP], it also includes the integrated circuit, the operating system that interfaces the Java Card Runtime

[5] This service is explicitly excluded from the evaluation scope this Security Target; see Section 3.2.1.

Environment with the chip, and the Card Manager enabling to enlarge or reduce the set of applications that the card offers to the Cardholder.

Any native application that could be embedded in the chip is out of the scope of the TOE. Native applications are considered as being part of the TOE IT environment. This Security Target assumes that they are harmless with respect to all the security policies of the platform.

The Java Card applets are also excluded from the scope of the TOE, because they are considered as data managed by the TOE. This means that any application-specific TSF is out of the scope of this Security Target. Moreover, the requirements in this Security Target do not span (actually, they do not need to span) all the stages in the development cycle of a Java Card application. Applets are only considered in their CAP format, and the process of compiling the source code of an application and converting it into the CAP format does not regard the TOE or its environment. On the other hand, the processes of verifying CAP files and loading them on the card are a crucial part of the TOE environment and play an important role as a complement to some of the on-card security functions. For this reason, this Security Target requires the enforcement of organizational security policies regarding those activities, and imposes security functional requirements on the implementation of the bytecode verifier.

The TOE includes the part of the Java Card Application Programming Interfaces that the platform supports plus some additional (proprietary) Java Card libraries developed by Trusted Logic. These proprietary libraries are:

- `fr.trustedlogic.util`
- `fr.trustedlogic.security`

The TOE has been designed to support a configurable number of logical channels, which can be set up during the initialization phase of its manufacturing process. For the sake of the evaluation, it is assumed that the Card Manufacturer initializes the TOE so that one single logical channel can be opened at most. Similarly, although the TOE does support Remote Method Invocation from the terminal, this mechanism is excluded from the scope of evaluation. This means that that the Card Administrator is expected to deny the loading of any applet relying on this mechanism.

Regarding the TOE's life cycle, this Security Target only covers the development of the software to be embedded on the IC and those states once the TOE has become operational, this is, once the code of the Java Card System can be executed. The construction of the IC and the smart card and embedding process itself are addressed in [ICST] and are out of the scope of this Security Target.

3.2.2. Users and Roles

The users of the TOE include the following people and institutions.

Platform Manufacturer

The Platform Manufacturer is a generic term that includes all the actors involved in the Platform Development phase of the smart card's life cycle. During this phase, the role of the Platform Manufacturer is in turn embodied by the Application Provider, the Platform Developer, the IC Manufacturer, and the Card Manufacturer.

Platform Developer

The Platform Developer is the organization responsible for designing and implementing the software masked on the IC. This includes the following components of the TOE: Card Manager, Java Card Runtime Environment, and Operating System.

Application Provider

An Application Provider is an organization that develops Java Card applications on demand of the Card Issuer. These applications implement services that the Card Issuer proposes to the Cardholder.

IC Manufacturer

The IC Manufacturer integrates the Embedded Software within the IC. This is usually known as the "masking" process.

Card Manufacturer

The Card Manufacturer integrates the masked IC with the carrier (a plastic card, a passport booklet, etc) in accordance with the Card Issuer's requirements, to produce a complete smart card ready for delivery to the Card Enabler.

Card Enabler

The Card Enabler is responsible for preparing the smart card for platform initialization according to the instructions of the Card Issuer. It may load configuration data into the smart card and also load or replace the static ISD keys to be used for the authentication of the Card Administrator.

Card Administrator

The Card Administrator is the representative of the Card Issuer that has ultimate control of the smart card's content and life cycle management. During the platform initialization phase, this role is embodied by the Card Enabler. During the platform usage phase, the Card Administrator can lock, unlock or terminate the smart card, download new applets on it, modify the static keys of the ISD or retrieve administration information from the smart card. The Card Administrator always acts on behalf of the Card Issuer.

Verification Authority

The Verification Authority is in charge of ensuring that the Java Card applets to be installed on the smart card do not violate any of Card Issuer's security policies. In particular, the Verification Authority is responsible for the bytecode verification of the downloaded applets, as well as for checking that the Application Developer did respect all the security recommendations specified in **¡Error! No se encuentra el origen de la referencia.**.

Cardholder

The Cardholder is the person or group of persons that the Card Issuer designs as the rightful holder of the smart card.

Card User

A Card User is any person presenting the smart card to the terminal and claiming the identity of the Cardholder.

3.2.3. *The TOE in the Smart Card's Life Cycle*

The life cycle of the TOE is compliant with the standard [VCPG] and refines the one described in [JCSPP]. Figure 2, taken from [CSRS] presents its main states. They are organized into two phases, named Platform Production and Platform Usage. Those phases are detailed in the sequel.

3.2.3.1. Platform Production

The Platform Production phase is in turn composed of two sub-phases: Platform Development and Platform Initialization.

Platform Development

Platform Development starts by the design of the IC and all the components of the Embedded Software: Operating System, Java Card Runtime Environment, Card Manager and the collection of built-in applications to be masked with the code of the platform. For the development of the Operating System, the Platform Developer uses tools and manuals provided by the IC Manufacturer. For the development of the Java Card Runtime Environment and the Card Manager, the Platform Developer uses the specifications of those standards provided by SUN Microsystems, GlobalPlatform and VISA.

Platform Initialization

The IC Manufacturer then masks the Embedded Software on the IC. To do this, the Platform Developer provides the IC Manufacturer with the binary files of the Embedded Software and all the configuration files to be written in EEPROM. The result of the masking process is an IC containing the Embedded Software, called a *module*.

The module is then booted for the first time and it enters a state where it is ready for receiving initialization APDU commands. If necessary, the IC Manufacturer may load a patch for the Embedded Software as part of the initialization process. The patch file is developed and provided by the Platform Developer and can only be applied at this life cycle state.

The IC Manufacturer then delivers the modules to the Card Manufacturer, which embeds each module into its plastic or paper carrier. The result of this process is a complete smart card that is ready for receiving personalization commands.

3.2.3.2. Platform Usage

Once the platform has been successfully initialized, the Card Enabler initializes the Issuer Security Domain with the Card Issuer's data. The Card Issuer provides the Card Enabler with the keys and other initial data required for personalizing the platform, as specified in [VCPG].

After being initialized with the Card Issuer's data, the Card Issuer has complete control of the TOE and the smart card irreversible enters the Operational Phase. At this point, all the security functions defined in this Security Target are activated and operational. The Card Issuer then chooses the set of Application instances to be installed on the card and the Card Administrator loads and personalizes each one with the Cardholder's data. For developing the applications ordered by the Card Issuer, the Application Provider uses the specification of the Java Card Application Programming Interface provided by SUN Microsystems and the user guides provided by the Platform Developer. For loading new applications and personalizing them with the Cardholder's data, the Card Administrator uses the administration guides provided by the Platform Developer.

After being initialized with the Cardholder's data, the TOE is delivered to the Cardholder. The Card Issuer provides the Cardholder with user guides for accessing to the services provided by the smart card in the most secured way. During this stage, the Card Administrator also performs all the card management activities described in Section 3.1.5 following the same administrator guides provided by the Platform Developer for the personalization step. This includes downloading new applets on the smart card, according to the instructions of the Card Issuer.

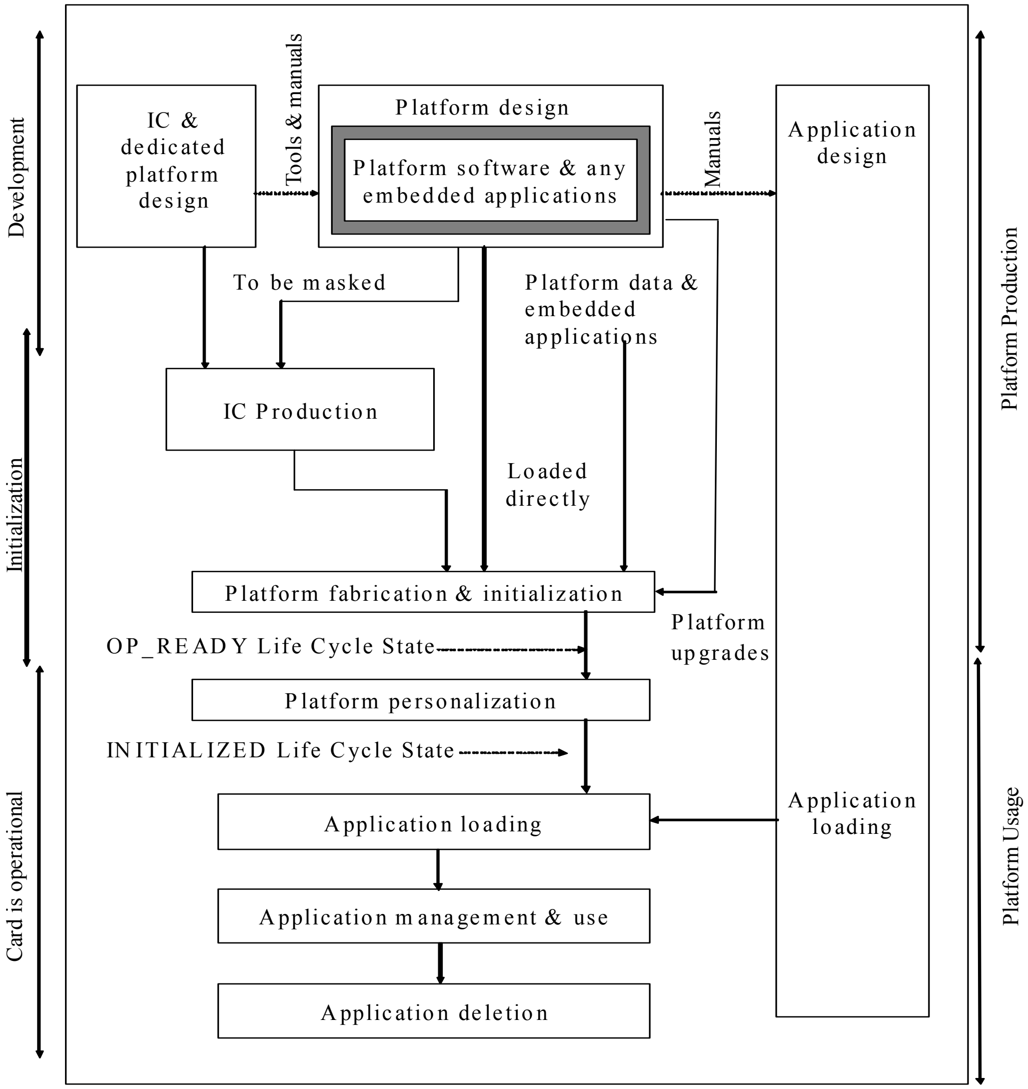

Figure 2: Development and operation lifecycle

The scope of this Security Target only includes the Platform Usage phase of the smart card's life cycle after the Platform Personalization step (once the card has reached at least the INITIALIZED state) and a single step of its Development Phase, namely, the development of the Embedded Software. All the other phases are beyond of the scope of this Security Target.

3.3. TOE Intended Usage

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

The TOE 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 interacts with a card reader. The Card Issuer may also use the card reader to enlarge or to restrict the set of applications that can be executed on the Java Card platform, or to enforce other card management policies, like personalizing the application instances or modifying the life-cycle state of either the whole card or one of its application instances. Only the Card Administrator is supposed to perform card management operations: in particular, neither the Cardholder nor any Application Provider is supposed to download Executable Files to, or remove them from, the card without the explicit authorization and participation of the Card Issuer. The Cardholder is only authorized to make use of the services proposed by the application instances that the Card Issuer has created for him or her.

So far, the most important applications concern:

- 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;
- Telephony, through the subscriber identification module (SIM) for digital mobile telephones;
- Electronic signature devices for e-government, like applications based on the standard described in [SSCD];
- Electronic passports and other machine-readable transport documents, like those based on the standard described in [MRTD];
- Personal identification for granting access to secured sites or providing identification credentials to participants of an event;
- Secure information storage, like health records, or health insurance cards;
- Loyalty programs, like the "Frequent Flyer" points awarded by airlines.

Application instances may therefore contain other confidentiality or integrity sensitive data than usual cryptographic keys and PIN codes; for instance: passwords, pass-phrases, or personal information like biometric data, health-care information or the balance of an electronic purse are as confidential as the PIN. Such highly sensitive information may co-exist with other kind of information, like the number of points of a fidelity program.

4 TOE security environment

4.1. Assets

This section introduces the assets that the TOE shall protect.

Assets may overlap, in the sense that different assets may refer (partially or wholly) to the same piece of information. 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 into the assets under the control of the Card Manager and the assets under the control of the Runtime Environment. Each group of assets is classified in turn according to whether it is data created by and for the user of the TOE (User data) or data created by and for the TOE (TSF data). The kinds of dangers that weigh on each asset are also specified.

4.1.1. Card Management Assets

This section introduces the assets under the control of the Card Manager.

4.1.1.1. User Data

D.APP-CODE

Application code

The code of the applications and libraries that is executed by the Runtime Environment.

The on-card executable code of a single application is called an Executable Module. The actual on-card container of one or more Executable Modules is called an Executable File. Before being installed on the card, an Executable File is called an (Executable) Load File.

In Java Card Terminology, an Executable Module is called an Applet class. The container of one or several Applet classes is called a CAP (Converted APplet) file, or a Package. The word *bytecode* is used to refer to the instructions for the Runtime Environment that are contained in a CAP file.

This asset shall be protected from unauthorized modification.

D.APP-INST

Application instance

An instance of an Executable Module of one of the Executable Files loaded on the card.

Each application instance provides a collection of services to the users of the smart card.

This asset shall be protected from unauthorized modification.

D.COMMAND

APDU command

The APDU commands that the Issuer Security Domain accepts.

An APDU command addressed to the ISD contains a request for a card management service. Valid requests come either from the Cardholder or from the Card Administrator.

This asset shall be protected from unauthorized modification. Some specific card management commands, like those containing keys, shall be also protected from unauthorized disclosure.

4.1.1.2. TSF Data

D.ISD-SESSION-KEYS

Session keys

The cryptographic keys that the ISD uses for ensuring the integrity and origin of card management requests.

This asset shall be protected from unauthorized disclosure and modification.

Application note:

This asset is named D.JCS_KEYS in [JCSPP]. It was renamed in this Security Target in order to highlight that in GlobalPlatform those keys are actually held and managed by the ISD. Notice that the static keys used to derive the session keys are included in the D.ISD-PERSO asset.

D.ISD-PERSO

Card personalization data

The Card Issuer's data used to personalize the smart card.

The Issuer Security Domain (ISD) is the on-card representative of the Card Issuer. As such, it stores cryptographic keys needed to support several card management functions, like setting up a secure channel with the terminal.

These assets shall be protected from disclosure and unauthorized modification.

D.GP-REGISTRY

GlobalPlatform's Registry

The information that the OPEN keeps about the Executable Load Files and the installed application instances.

The GP Registry contains the following information about each application instance:

- identifier,
- privileges,
- current life cycle state,
- memory resource quotas,

The current life cycle state of the whole smart card is also part of the GP registry.

This asset shall be protected from unauthorized modification.

D.CVM

Cardholder Verification Method

The data associated with the Cardholder Verification Method (CVM). The CVM is a single global PIN used to authenticate the Cardholder, which can be shared by all the application instances of the card.

The CVM data includes the following items:

- the PIN code used to authenticate the Cardholder,

- the current number of failed authentication attempts,
- the maximum number of authentication attempts.

These assets shall be protected from unauthorized modification. The PIN code of the CVM shall be also protected from unauthorized disclosure.

4.1.2. Runtime Environment Assets

This section introduces the assets under the control of the Runtime Environment. These assets are those introduced in [JCSPP], with the exception of the D.APP-CODE, which was already introduced in the previous section.

4.1.2.1. User Data

D.APP_C_DATA

Confidential application data

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

This asset shall be protected from unauthorized disclosure.

D.APP_I_DATA

Integrity-protected application data

Integrity sensitive data of the applications, like the data contained in a Java Card object, a static field of a package, a local variable of the currently executed method, or a position of the operand stack of the JCVM.

This asset shall be protected from unauthorized modification.

D.PIN

Application PIN code

Any PIN code that an applet uses to authenticate the Cardholder and its security attributes (try counter and limit). This value may be different from the global CVM under the control of the Card Manager.

These assets shall be protected from unauthorized modification. The PIN value shall also be protected from unauthorized disclosure.

D.APP_KEYs

Application keys

The cryptographic keys owned by the application instances.

This asset shall be protected from unauthorized disclosure and modification.

4.1.2.2. TSF Data

D.SOFTWARE

Embedded Software

The instructions that compose the Embedded Software.

The Embedded Software includes the code masked in ROM and the Patch File loaded in the persistent mutable memory of the card during its initialization phase.

This asset shall be protected from unauthorized disclosure and modification.

Application note:

This asset covers the D.JCS_CODE asset in [JCSPP]. The term *the code of the Java Card System* used in that protection profile has been replaced in this Security Target by the broader term *Embedded Software*, which also includes the code of the Operating System and the code of the Card Manager.

D.JCS_DATA

Runtime data areas

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

To be protected from monopolization and unauthorized modification.

D.SEC_DATA

Sensitive data of the Runtime Environment

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

This asset shall be protected from unauthorized modification.

D.API_DATA

APIs private data

Private data of the API, like the contents of the private fields of the Java Card classes of the JC and the GP APIs.

This asset shall be protected from unauthorized disclosure and modification.

D.CRYPTO

Cryptographic data

Any kind of cryptographic data used in runtime cryptographic computations that is different from a cryptographic key, like a seed used to generate a key, or the result of a hashing function.

This asset shall be protected from unauthorized disclosure and modification.

4.2. Subjects

This section introduces the subjects and users involved in the use of the TOE. The aim is to put the subjects introduced in [JCSPP] in correspondence with the ones mentioned in this Security Target.

A subject is an active component that interacts with the TOE on behalf of a user. Following the classification described in the Common Criteria [CC2], the users of the TOE are divided into two categories: human users and IT entities. Human users of the TOE include people or institutions, like the Cardholder, the Card Issuer or an Application Provider. IT entities include hardware and software components, like a terminal or an application instance.

4.2.1. *IT Entities*

The IT entities are programs that invoke the services offered by the platform. This includes programs running on the CAD's side on behalf of the Card Administrator as well as applets running on top of the Java Card platform on behalf of the corresponding Application Provider.

4.2.1.1. Subjects associated to the Card Manager

The following subjects are involved in the security policies enforced by the Card Manager.

S.APP

An *application instance* is an on-card entity implementing one of the services offered by the smart card, possibly using the services that GlobalPlatform offers through its API.

The actual set of applications embedded on the card depends on each card and can be dynamically enlarged.

This subject is called S.PACKAGE in [JCSPP]. This name comes from the fact that the Java Card Firewall security policy grants the same privileges to all the applet instances declared in the same package.

S.NAT

A *native application* is an application written in the native language of the platform. A native application is not executed through the Runtime Environment.

S.OPEN

The *OPEN* is the embedded software component in charge of dispatching the APDU commands to the Application instances. This subject acts on behalf of the Card Issuer.

S.SPY

The *Spy* is any subject acting on behalf of the attacker with the purpose of disclosing, replaying, deleting, modifying or permuting the APDU commands sent to card.

S.ISD

The *Issuer Security Domain* is a distinguished application instance in charge of enforcing the Card Issuer's security policies on the card.

According to GlobalPlatform's specifications, this subject embodies the roles and responsibilities of the following subjects introduced in [JCSPP]:

- S.CRD (applet installer)
- S.ADEL (applet deletion manager)

4.2.1.2. Subjects associated to the Runtime Environment

The following subjects introduced in [JCSPP] are involved in the security policies of the Runtime Environment.

S.JCRE

The *JCRE* stands for those software libraries of the TOE written in Java Card and which are executed on behalf of the Card Issuer. This subject is involved in the Java Card Firewall policy of the Runtime Environment.

S.BCV

The *Bytecode Verifier* is a program that verifies the code of the Executable Load Files downloaded on the card. It acts on behalf of the Verification Authority. This subject is involved in the Card Content Management security policy.

4.3. Assumptions

This section describes the assumptions that are made regarding the TOE security environment. They are classified into assumptions from [JCSPP] concerning the Embedded Software and assumptions from [ICST] concerning the Integrated Circuit. The TOE of those documents is different from that of this Security Target, so the set of assumptions has to be suitably adapted. On one hand, some of the assumptions in those documents are either covered by weaker assumptions of this Security Target, or discharged (represented) by its threats or requirements. In this latter case, it is up to the TOE to counter that threats or fulfill those requirements, so there is no need to rely on the assumption. On the other hand, the assumptions in [JCSPP] and [ICST] do not concern proprietary features of the TOE. Consequently, this Security Target also introduces some extra assumptions concerning those features.

4.3.1. Assumptions on the Embedded Software

The scope of the Embedded Software addressed in this Security Target has been enlarged with respect to [JCSPP] so as to include the Card Manager. As a consequence, the assumption A.CARD-MANAGEMENT included in that protection profile is no longer pertinent, and it is discharged by the threats considered in the following section. The A.NATIVE assumption has been also adapted to the TOE defined in this Security Target, which includes the whole Java Card API, including native methods.

Some extra assumptions about the behavior of the actors involved in the card manufacturing and card administration processes are also included to ensure that the security objectives defined in this Security Target are sufficient for countering those broader threats.

A.NATIVE

All pre-issuance native application on the card are assumed to be compliant with the TOE so as to ensure that security policies and objectives described herein are not violated.

Application note:

In [JCSPP], this assumption also supposes that native methods of the Java Card API also enforce the security policies defined for the platform. This part of the assumption has been discharged in this Security Target, as the native libraries of the Operating System that support API implementation are also in the scope of this TOE.

A.VERIFICATION

All the bytecodes of the Executable Files masked on the card have successfully passed the Bytecode Verification process and have not been modified after being verified. Moreover, they only contain applets that follow the security recommendations stated in [USR].

A.APPLET

The Executable Files loaded in the post-issuance phase do not contain native methods. The Java Card specification explicitly does not provide any support for native methods apart from those of the Java Card API.

A.MANUFACTURING

The Platform Manufacturer is supposed to ensure the quality, confidentiality and integrity of the manufacturing process as well as the disabling of all test and debug mechanisms from the product once the smart card is in the OP_READY state (at the beginning of the Platform Personalization phase). The Platform Manufacturer and the Card Enabler are assumed to use only proper secret keys (chosen from a large key space) as input for the cryptographic functions of the TOE.

It is assumed that both the Card Enabler and the different actors embodying the role of the Platform Manufacturer (Application Providers, Platform Developer, IC Manufacturer, Card Manufacturer) do enforce security procedures to maintain confidentiality and integrity of the TOE and of its manufacturing and test data up to delivery to the Cardholder. In particular, the TOE is assumed to be protected adequately when it is delivered from one actor to the other, so as to prevent its copy, modification, retention, theft or unauthorized use.

4.3.2. Assumptions on the IC

The scope of the TOE addressed in this Security Target has been enlarged with respect to [ICST] so as to include the Embedded Software. In addition to this, the part of the life cycle under evaluation has been both restricted (manufacturing of the IC is excluded) and enlarged (some of the phases after IC delivery are included). As a consequence, the assumptions included in [ICST] are not longer pertinent for the TOE addressed in this Security Target, as explained below.

The following assumptions from [ICST] are covered by weaker ones included in this Security Target:

- A. Process-Card, which assumes that the IC is protected after the IC Manufacturer delivers it to the other actors, is covered by the weaker assumption A.MANUFACTURING, which includes all the roles involved in the Platform Development phase: Platform Developer, Application Provider, IC Manufacturer, Card Manufacturer.

The following assumptions from [ICST] have been discharged:

- A.Resp-App, which assumes that the Embedded Software protects the keys and other sensitive data, is discharged by the threats considered in the following section, especially by T.CRYPTO, T.CONFID-APPLI-DATA, T.CONFID-JCS-DATA, T.INTEG-APPLI-DATA, T.INTEG-JCS-DATA.
- A.Plat-Appl, which assumes the correct design of the Embedded Software according to IC manufacturer requirements, is discharged by the assurance measures concerning the development of smart card embedded software and its environment.
- A.Key-Function, which assumes that the Embedded Software protects the cryptographic keys from information leakage, is also discharged by the threats considered in the previous section, specially by T.CRYPTO.

4.4. Threats

This section describes the threats on the assets against which specific protection within the TOE or its environment is required.

Each threat is introduced giving its reference, a short description of the general method used by the attacker, and possibly some attack scenarios. Threats are formulated so that they are as disjoint as possible.

Each threat is always associated with the assets that are directly threatened and with the subjects that may be involved in the attack, notably the subject that could perform the attack. Such information is not displayed in the definition of the attack, but in the TOE rational associated to each threat.

The threats are grouped into those menacing the assets related to card management operations, those regarding the assets protected by the runtime environment, and those concerning the underlying hardware. Those threats concerning the runtime environment come from [JCSPP], while those concerning the underlying hardware come from [ICST]. Most of the threats related to card management are specific to this Security Target, but have been largely inspired from [CSRS] and [JCSPP].

4.4.1. Card Management

This section introduces the threats concerning card management. The general description of the threat is sometimes followed by examples illustrating particular scenarios for it.

Those threats in [JCSPP] that directly concern card management operations are also included in this section, see T.INSTALL and T.DELETION.

T.IMPERSONATE

The attacker may try to impersonate the Cardholder in order to gain access to the services that the card offers to him.

Impersonating the Cardholder amounts to disclosing or guessing the PIN code stored in the CVM.

T.INVALID-INPUT

The attacker may determine security relevant information, cause the card to malfunction or otherwise compromise security through introduction of invalid inputs.

Invalid input may take the form of operations that are not formatted correctly, requests for information beyond register limits, or attempts to search for possible undocumented commands. Such inputs could be generated at any time during the normal usage of the card, including prior to access authorization, through normal operations. The attack could also make use of invalid data and inappropriate operations, such as commands/functions with requests/formats that are out of range or otherwise non-conforming to the *accepted* usage. The result of such attacks may be a compromise in the security functions, generation of exploitable errors in operation, or release of protected data.

Application note:

An invalid parameter is a piece of data that is not encoded in the expected format, or which has been forged by the attacker by other means that those defined in the functional specification of the TOE. An invalid parameter represents a value that should never be used according to the implementation chosen for a particular type of data.

The following are examples of possible invalid parameters concerning security sensitive information:

- a command with one of the INS bytes defined in [GPCS] which does not have the expected values in the P1 and P2 parameters, or the expected TLV structure in its data filed;
- a key of an invalid length;
- a reference to a component of the key which does not exists for the given key;
- a cryptographic algorithm that is not supported by the platform;
- a message to be encrypted or signed that is not correctly aligned with respect to the cryptographic algorithm to be used;
- a PIN code in hexadecimal format that is interpreted as if it was in decimal form;
- a PIN code of an invalid length;
- a secure channel protocol that does not correspond to any of the protocols that the TOE implements, like for instance the SCP02 protocol introduced in GlobalPlatform 2.1.1;
- a byte-encoded record structure which does not respect the expected encoding format, like a byte where some bits are expected to be set with a specific value.

T.INVALID-ORDER

The attacker may corrupt the internal data structures of the platform through the invocation of the functions provided by the interface in an unexpected order.

This threat includes the possibility for a friendly user to invoke the functions implementing a TSF in an order that compromises the security measures they enforce. For instance, the programmer of an application could try to encrypt a piece of information before specifying the cryptographic algorithm to be used. It also covers the possibility that the attacker calls a TSF before this function has been correctly created and has initialized its internal data structures, exploiting the security breaches exposed by an incoherent or outdated state of the TSF.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The attacker may send to the card a suite of commands that is not in the expected order, like for example:
  - a LOAD command that is not preceded by an INSTALL one,
  - a sequence of STORE DATA commands personalizing the ISD data where the commands are not arranged in the expected order,
  - a SELECT command sent to an application that has not been installed yet.
- The attacker may require performing an authentication operation before all the parameters required by the authentication service have been supplied. For instance, the attacker may try to be authenticated as the Cardholder before the CVM service has been initialized with a PIN code.
- The attacker may try to consult the value of a key after the end of its lifetime.

T.FORCED-RESET

The attacker may force the card into an insecure life cycle state through inappropriate termination of selected operations.

Attempts to generate a non-secure life cycle state in the card may be made through premature termination of transactions or communications between the card and the card reading device, by insertion of interrupts, or by selecting deleted applications that were not able to remove its objects. An attacker may also corrupt security sensitive data by provoking the interruption of the execution of the TSF. This includes tearing the smart card from the CAD as well as firing any other mechanism that could stop or deviate the normal execution of a TSF, like hardware interruptions, input/output interruptions, etc.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The attacker tires to interrupt the execution of an unsuccessful PIN-code verification before the number of remaining attempts has been updated, so that this value is actually never decremented.
- The attacker tries to interrupt the loading of one of the secret keys of the card.
- The attacker tries to interrupt the copy of a secret key into the buffer used by hardware cryptographic functions, in order to shrink the length of the key during an encryption operation.

T.REPLAY

The attacker may penetrate on-card security through the reuse of a completed (or partially completed) operation that was previously performed by an authorized user.

A completed (or partially completed) operation may be replayed in an attempt to bypass security mechanisms or to expose security-related information. For instance, the attacker may try to send to the card an APDU command that he intercepted in a previous session.

The attacker may also use authentication information that was previously delivered to him in order to disclose or modify a piece of information stored in the card that is currently in use by other applications. For instance, the attacker may use authentication information that was once valid, but that is not longer valid, like an old PIN value or cryptographic key.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The aim of the attacker may be, for example, to perform the deletion of an application instance, or the loading of malicious applications. An example of a sequence of commands that could be intercepted for replay is the original INSTALL[for load], LOAD and INSTALL[for install] commands used to create an application instance that has been deleted. If the application had been deleted because it contains security vulnerabilities, the reloading might enable the attacker to continue to exploit that vulnerability.
- The attacker tries to use a previous session key in order to decrypt or falsify a message sent to the card.
- The attacker tries to intercept a command containing a secret key to be loaded on the card, guess the key by some means, and replay the intercepted message later on in order to set the broken key again.
- The attacker tries to re-play an outdated PIN code that was formerly used by the Cardholder.

T.BRUTE-FORCE

The attacker may search the entire user-accessible data space to identify platform and application data.

APDU commands (API methods) can be repetedly transmitted (invoked) to attempt the brute force extraction of secrets such as cryptographic keys or PINs. This threat is distinguished by the use of valid commands with valid range requests that are repeated to reveal as much as possible of the data space. For example, an attacker may systematically experiment with different forms of input. The attack is based on the black box software engineering technique of establishing the nature of algorithms and predicates. If carried out exhaustively it could facilitate the reverse engineering of particular applications as well as the extraction of operational and security related information. The attack could also generate errors in the operation of the card. It may make use of the same valid command with valid range requests repeated many times.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The attacker may search the entire space of keys, signatures or message authentication codes to decrypt, falsify or silently modify a piece of data.
- The attacker may search the entire space of PIN codes in order to detect the one identifying the Cardholder.

T.LIFE-CYCLE

The attacker may exploit commands that were necessary for another part of the card life cycle but are not currently allowed, to expose TSF data or sensitive application data.

Certain card management commands may not be required or allowed in the specific phase of operation being executed. Examples include use of commands which have no operational use, but which could be used to test or debug the internal interfaces of the systems implementing the TOE.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The attacker tries to load an Executable File before the keys required for opening a Secure Channel with the terminal have been populated.
- The attacker tries to apply a patch to the code of the TOE after card initialization;
- The attacker tries to modify the card contents when the card has been temporarily locked;
- The attacker tires to execute an application instance that has been temporarily disabled or definitely deleted.

This attack covers the T.Abuse-Func introduced in [SSVG]: *An attacker may use functions of the TOE which may not be used after TOE Delivery in order to (i) disclose or manipulate User Data, (ii) to manipulate (explore, bypass, deactivate or change) security features or functions of the TOE or of the Smartcard Embedded Software or (iii) to enable an attack.*

T.INSTALL

The attacker may fraudulently install an Executable File or application instance on the card.

This threat concerns either the installation of bad-formed or ill-typed code, or an attempt to induce a malfunction in the TOE through the installation process, for instance by corrupting the Executable File during its installation. It also concerns the attribution of abusive privileges during the activation of an Executable Module that has been licitly loaded on the card.

If the card has been set up to the closed state, this threat also concerns any attempt from the attacker to load additional Executable Files in the card or creating new applet instances.

T.DELETION

The attacker deletes an installed Executable File or application instance that is already in use by some other file or application instance on the card, or uses the delete functions to pave the way for further attacks by putting the TOE in an insecure state.

Insecure states could be the result of broken references to garbage collected code or data, leading to information containers that have been reused by the platform for other purposes.

The unauthorized deletion of applications could be intended as a denial of service attack, as part of an attack directed at the Issuer Security Domain back-end systems or in attempt to delete evidence of another attack.

Application note:

The following list illustrates some possible scenarios for this kind of attack:

- The attacker tries to delete an application instance that has shared some of its class instances with another application instances;
- The attacker tires to delete an Executable File which contains a library of the JCAPI or the GPAPI that is necessary for the correct execution of the TOE;
- The attacker tires to delete the ISD;
- The attacker tries to delete an Executable File in order to free its allocated memory blocks, and then try to gain access to the information contained in that block through the allocation functions.

4.4.2. Runtime Environment

This section introduces the threats to the assets under the control of the Runtime Environment. Several groups of threats are distinguished according to the means used in the attack. The classification is also inspired by the components of the TOE that are supposed to counter each threat.

All the threats introduced in this section come from [JCSPP].

4.4.2.1. Confidentiality

T.CONFID-JCS-CODE

The attacker executes an application without authorization to disclose the code of the Embedded Software.

This threat 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 platform code is stored.

Application note:

The term *Java Card System* used in [JCSPP] has been replaced in this Security Target by the broader term *Embedded Software*, which also includes the Card Manager and the Operating System.

T.CONFID-APPLI-DATA

The attacker executes an application to disclose data belonging to another application without its authorization.

This concerns logical attacks at runtime in order to gain read access to the class instances and the arrays created by the other application instances.

T.CONFID-JCS-DATA

The attacker executes an application to disclose private data belonging to the Embedded Software.

This concerns logical attacks at runtime in order to gain read access to the private data of the Runtime Environment, the Operating System or the Card Manager. Private data of the Runtime Environment includes TSF data contained in the runtime data areas of the Java Card Virtual Machine and the private fields of the classes implementing the Java Card API.

Application note:

The term *Java Card System* used in [JCSPP] has been replaced in this Security Target by the broader term *Embedded Software*, which also includes the Card Manager and the Operating System.

4.4.2.2. Integrity

T.INTEG-APPLI-CODE.1

The attacker executes an application to alter (part of) its own or another application's code.

This concerns logical attacks at runtime in order to gain write access to the memory zone where executable code is stored.

T.INTEG-JCS-CODE

The attacker executes an application to alter (part of) the Embedded Software.

This concerns logical attacks at runtime in order to gain write access to executable code.

Application note:

The term *Java Card System* used in [JCSPP] has been replaced in this Security Target by the broader term *Embedded Software*, which also includes the Card Manager and the Operating System.

T.INTEG-APPLI-DATA.1

The attacker executes an application to alter (part of) another application's data.

This threat concerns logical attacks at runtime in order to gain unauthorized write access to application data.

T.INTEG-JCS-DATA

The attacker executes an application to alter (part of) the internal data structures of the Embedded Software or the private data of the JCAPI.

This concerns logical attacks at runtime in order to gain write access to the private data managed by the Java Card Runtime Environment, the Java Card Virtual Machine and the private data of Java Card API classes.

Application note:

The term *Java Card System* used in [JCSPP] has been replaced in this Security Target by the broader term *Embedded Software*, which also includes the Card Manager and the Operating System.

T.INTEG-APPLI-CODE.2

The attacker modifies (part of) its own or another application code when an application package is transmitted to the card for installation.

This threat concerns the modification of application code in transit to the card.

T.INTEG-APPLI-DATA.2

The attacker modifies (part of) the initialization data contained in an Executable Load File when it is transmitted to the card for installation.

This threat concerns the modification of the values to be used for initializing the static fields defined in the Executable Load File. It also covers the personalization data that the Card Administrator sends to an installed applet during applet personalization.

Application note:

This objective enlarges the corresponding one in [JCSPP] by taking into account the data used during the applet personalization phase.

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

4.4.2.3. Identity Usurpation

T.SID.1

The attacker either impersonates an on-card subject or an external user of the card.

In order to impersonate an on-card subject, the attacker may execute an applet instance which impersonates another applet instance, or even the JCRE. In this case, the attacker's aim is to gain illegal access to the internal resources that the Runtime Environment provides. The attacker may also try to impersonate an external user of the card, like the Card Administrator.

This attack concerns the fraudulent adoption of special privileges only granted to the Runtime Environment or to one of the applet instances registered on the card, and specially the default and the currently selected ones. It also concerns the possibility of confusing an external user of the smart card, who may think that he is communicating with one of his on-card representatives, while he is actually communicating with the attacker.

Application note:

This threat refines the corresponding one in [JCSPP] by specifying the *resources* that can be accessed and the *end users* that the attacker could try to impersonate.

T.SID.2

The attacker modifies the identity of the privileged roles.

This threat concerns the fraudulent adoption of a privileged role, like the Default or Selected Applet. The actors embodying these roles have access to platform resources or services that may not be available to other applet instances. The attacker aims to get unauthorized access to those resources and services.

Application note:

The following list illustrates some possible scenarios for this threat:

- The attacker tires to change the life cycle state of another applet or the whole card, to modify the card contents or to perform any other operation which is only allowed to the Card Administrator.
- The attacker executes an application that fraudulently tries to lock or terminate the card without authorization.
- The attacker executes an application that tries to reset the PIN code of the CVM.
- The attacker tries to set a malicious application instance as the default selected one, so that it catches commands that are intended for other application instances.

This threat details the corresponding one in [JCSPP] by stating that the privileged roles that the attacker could try to modify are the ones defined for an applet in [VGP].

4.4.2.4. Unauthorized Execution

T.EXE-CODE.1

The attacker executes an applet instance that performs an unauthorized execution of a method.

This threat concerns:

- invoking a method outside the scope of the visibility rules provided by the public/private access modifiers of the Java programming language;
- invoking a method on class instance that its owner did not declare as being shareable with the invoker of the method

T.EXE-CODE.2

The attacker executes an applet instance that performs an unauthorized execution of a method fragment or arbitrary data.

This attack concerns jumping inside a method fragment, or interpreting the contents of a data memory area as if it was executable code.

T.NATIVE

The attacker executes a native method to bypass a security function such as the Java Card Firewall.

The execution of native code is not under the control of the Java Card Virtual Machine, so it must be secured to prevent bypassing the TSFs. No distrusted native code may reside on the card.

4.4.2.5. Denial of Service

T.RESSOURCES

The attacker prevents correct operation of the Java Card System through the monopolization of the card's RAM or NVRAM resources.

This attack concerns loading and executing an applet instance that could monopolize all the available smart card memory, so that access to other card services is systematically denied.

4.4.2.6. Cryptography

T.CRYPTO

The attacker may defeat the TSFs through a cryptographic attack against the algorithm or through a brute-force attack on the function inputs.

This attack involves any cryptographic function including encode/decode functions, signature functions or random number generators. The attacker's goal is either to exploit a weakness in the algorithm itself or in the way it was implemented, or, through brute-force substitutions, to find the appropriate keys and inputs. Weaknesses in cryptographic algorithms include both vulnerabilities raising from the theoretical ground on which the algorithm relies and weaknesses possibly involving information leakage during the operation with cryptographic keys that could be observed through SPA, DPA, DFA, or EMA techniques. The attacker's ultimate goal is to disclose sensitive platform or application data.

4.4.2.7. Services offered to the applets

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 crash, or to gain access to a memory containing data that is now being used by another application.

The aim of the attacker is to get access to services that the service provider believes no longer available, or to get unauthorized access to private information owned by other applets. For instance, the attacker could try to read the contents of a newly allocated array with the hope that it still contains residual information that was previously stored in a garbage-collected object. If the garbage collector introduces hanged references, the attacker could exploit those references to access a memory block being used by another application.

4.4.3. Integrated Circuit Threats

The following threats come from [ICST]. They refine the following generic threat in [JCSPP]:

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

As the TOE of this Security Target also includes the Embedded Software, the term *TOE* or *Smart Card* used in the threats of [ICST] has been replaced here by the more precise term *smart card's chip*. Some of the threats introduced in [ICST] have been gathered together for the sake of simplifying the presentation.

The threat T.ABUSE-FUNC in [ICST] is not included in this section, as it is already covered by the threat T.LIFE-CYCLE introduced above.

T.PHYS-TAMPER

The attacker may perform physical probing of the smart card's chip in order (i) to disclose TSF Data, or (ii) to disclose/reconstruct the smart card's chip Embedded Software.

An attacker may physically modify the smart card's chip in order to (i) modify security features or functions of the smart card's chip, (ii) modify security functions of the smart card's chip Embedded Software, (iii) modify User Data or (iv) modify TSF data. The physical tampering may be focused directly on the disclosure or manipulation of TOE User Data or TSF Data or indirectly by preparation of the TOE to following attack methods by modification of security features (e.g. to enable information leakage through power analysis). Physical tampering requires direct interaction with the smart card's chip internals. Techniques commonly employed in IC failure analysis and IC reverse engineering efforts may be used. Before that hardware security mechanisms and layout characteristics need to be identified. Determination of software design including treatment of User Data and TSF Data may also be a prerequisite. The modification may result in the deactivation of a security function. Changes of circuitry or data can be permanent or temporary.

This threat menaces all the assets.

Application note:

This threat gathers the threats T.Phys-Probing and T.Phys-Manipualtion in [ICST].

T.MALFUNCTION

The attacker may cause a malfunction of TSF or of the smart card's chip Embedded Software by applying environmental stress in order to (i) deactivate or modify security features or functions of the TOE or (ii) circumvent, deactivate or modify security functions of the smart card's chip Embedded Software.

This may be achieved by e.g. operating the smart card's chip outside normal operating conditions, exploiting errors in the smart card's chip Embedded Software or misusing the administration function. To exploit this, an attacker needs information about the functional operation.

This threat mainly menaces the Embedded Software code.

T.INF-LEAK

The attacker may exploit information leaked from the TOE during its usage in order to disclose confidential TSF data. The information leakage may be inherent in the normal operation or caused by the attacker.

Leakage may occur through emanations, variations in power consumption, I/O characteristics, clock frequency, or by changes in processing time requirements. This leakage may be interpreted as a covert channel transmission but is more closely related to measurement of operating parameters, which may be derived either from measurements of the contactless interface (emanation) or direct measurements (by contact to the chip still available even for a contactless chip) and can then be related to the specific operation being performed. No direct contact with the smart card's chip internals is required here. Examples are the Differential Electromagnetic Analysis (DEMA) and the Differential Power Analysis (DPA).

All assets are threatened, especially user data and cryptographic keys.

T.RND

The attacker may predict or obtain information about random numbers generated by the TOE, for instance because of a lack of entropy of the random numbers provided.

Here the attacker is expected to take advantage of statistical properties of the random numbers generated by the smart card's chip without specific knowledge about the smart card's chip generator. Malfunctions or premature aging may assist in getting information about random numbers.

This threat mainly concerns the session keys used to securely communicate with the Card Administrator's terminals.

4.5. Organisational security policies

This section describes the rules to which both the TOE and its human environment shall comply when addressing security needs.

4.5.1. Embedded Software OSP

This Security Target enlarges the organizational security policies introduced in [JCSPP] with some of the policies introduced in GlobalPlatform's specifications.

OSP.VERIFICATION

Before loading an Executable Load File on the card, the Verification Authority shall ensure that it respects the security recommendations stated in [USR]. In particular, the Verification Authority shall check that the Executable Load File successfully passes bytecode verification using Export Files that match the CAP files that are already installed on the card. Bytecode verification shall include:

- well-formedness of the CAP file structure and verification of the typing constraints on its bytecodes,
- binary compatibility with installed Executable Files and the assurance that the export files used to check the Executable Load File correspond to those that will be present on the card when loading occurs.

Upon successful verification of an Executable Load File, all the roles involved in card content management shall immediately activate all the IT and organizational measures required for preventing any modification of it until it is downloaded into the card. If the Card Manufacturer has configured the card to verify DAP signatures, then the Verification Authority shall electronically sign the file immediately after successful verification.

If the card has not been configured to verify DAP signatures, the Verification Authority shall transmit the Executable Load File to the Card Administrator through a secure communication channel ensuring the origin and the integrity of transmitted files. Upon reception the Card Administrator shall store the Executable File in its secure environment until the file is downloaded into the card. Obviously, this rule applies only when the Verification Authority and the Card Administrator are separate roles.

OSP.FILE-ORIGIN

Only the Card Administrator is allowed to transmit new Executable Load Files to the card.

If the card has not been configured to perform DAP verification, then the Card Administrator shall only download Executable Load Files received from the Verification Authority through the secure communication channel linking them. This rule only applies if the Verification Authority and the Card Administrator are separate roles.

OSP.SECRETS

Only the Platform Manufacturer and the Card Administrator may load secrets protecting the assets on the smart card, such us cryptographic keys and PIN codes. Such secrets shall be generated, distributed and stored off-card, destroyed and exported to the card in a secure manner, which prevents the attacker to obtain them from the IT or non-IT environment. PIN codes shall be transmitted to the Cardholder using a secure channel that ensures its origin, integrity and confidentiality. This rule also applies to DES keys that are not imported but automatically generated on-card by the applets. In this case it is up to the Verification Authority to check that such keys are securely generated, distributed, stored and destroyed.

The Verification Authority shall generate, store, and destroy the private key that it uses for DAP signature in a secure manner.

Application note:

PIN codes concern both the global PIN that the Card Manager implements as well as any specific PIN created by the installed applets.

OSP.KEY-LENGTH

The Verification Authority shall inspect the code of the applets and only validate those respecting the constraints regarding cryptographic key lengths provided in the Security Functional Requirements of this document.

Should an applet use the simple DES algorithm that the platform offers through the Java Card API, the Verification Authority shall ensure that the applet's algorithm chains several simple DES operations, so that to provide DES keys of at least 112 bits.

Application note:

The precise list of cryptographic algorithms defined in the Java Card API that fall into the scope of evaluation is summarized in the document [PROFILE].

OSP.NO-RMI-APPLETS

The Verification Authority shall not validate applets relying on the Remote Method Invocation (RMI) mechanism.

The Verification Authority is supposed to carefully inspect its bytecode, possibly with the help of static analysis tools, and to reject it if it contains a piece of code that relies on the following classes and interfaces of the Java Card API: `RMI`, `RMIService` and `CardRemoteObject`.

OSP.PERSONALIZATION

Until the card is successfully moved to a state where all the security functions are enabled, it shall be under the physical control of the Card Enabler, and shall be only used in a secure environment. Once the ISD is successfully transitioned to such state, the card shall be placed under the administrative control of the Card Administrator, who is the only role responsible for performing card content operations.

The card is issued to the Cardholder only after reaching the SECURED life cycle state described in GlobalPlatform's specifications.

4.5.2. Integrated Circuit OSP

As already explained in the assumptions, this Security Target enlarges the scope of [ICST] so as to include the Embedded Software. As a consequence, those organizational policies referring to the correct use of the interfaces of the IC by the Platform Developer are not pertinent for this Security Target, as they are fulfilled by the assurance measures of the class ADV (and in particular ADV_HLD). This is the case of the P.Add-Functions policy in [ICST]. The only pertinent organizational policy from [ICST] is therefore OSP.PROCESS-TOE.

OSP.PROCESS-TOE

The Platform Manufacturer and the Card Enabler must ensure that the smart card Platform Development and Platform Initialization phases are secure, so that no information is unintentionally made available for the operational phase of the TOE. For example, the confidentiality and integrity of design information and test data shall be guaranteed; access to samples, development tools and other material shall be restricted to authorized persons only; scrap will be destroyed, etc. This concerns all the actors embodying the role of the Manufacturer during the Platform Development phase: Application Provider, Platform Developer, IC Manufacturer and Card Manufacturer.

5 Security objectives

5.1. Security objectives for the TOE

This section defines the security objectives that the technical countermeasures implemented by the TOE shall satisfy. They are classified into three categories: the first one contains those objectives that are specific to card management; the second one those objectives introduced in [JCSPP] for the Runtime Environment; and the third one the objectives from [ICST]. Each objective is introduced giving its name and a short phrase summarizing it.

5.1.1. Objectives for the Card Manager

This section introduces the security objectives that are relative to card management.

5.1.1.1. Communication with the terminal

O.REQUEST

The TOE shall reject any card management request containing data that is not in the expected format.

In particular, those APDU commands which are ill-formed with respect to the functional specification of the TOE shall not be processed. Values of type short and byte which are outside of the expected range shall be also rejected by the methods of the JC and GP APIs.

O.INFO-ORIGIN

The TOE shall authenticate the origin of the card management requests that the card receives, and authenticate itself to the Card Administrator.

The goal is to ensure that the information modifying the security attributes of the card comes from a trustable actor who has carefully analyzed the consequences of the card management operation, namely, the Card Administrator. In the other way round, the on-card representative of the Card Administrator shall authenticate itself to that privileged user in order to prevent releasing sensitive information to a malicious applet.

Application note:

This objective generalizes the O.LOAD and O.INSTALL objectives in [JCSPP]. The verification of the origin applies to all card management operations, and not only to the loading of new Executable Files.

O.INFO-INTEGRITY

In the operational phase of the card, the TOE shall verify the integrity of the card management requests that the card receives.

The goal is to ensure that the card management operation processed by the card is exactly the one that the Card Administrator requested. In addition to this, each applications installed on the card may also request the ISD to verify the integrity of the commands used to personalize it. This only applies to the operational phase of the card, where the card is in a potentially hostile environment.

Application note:

This objective generalizes the O.LOAD and O.INSTALL objectives in [JCSPP] in the same way as O.INFO-INTEGRITY.

O.INFO-CONFIDENTIALITY

The TOE shall be able to process confidential requests containing encrypted data.

The goal is to prevent the disclosure of the secret keys enabling to open a Secure Channel between the ISD and the Card Administrator. In addition to this, the Application Providers may want to protect the code of their applications while they are in transit to the card. Finally, application instances may also request the ISD to provide a Secure Channel enforcing confidentiality for their own personalization.

O.NO-KEY-REUSE

The TOE shall ensure that session keys can be used only once.

The keys used to ensure the origin, integrity and confidentiality of the card management requests shall contain an unpredictable piece of data that is randomly chosen by the TOE, so that they can be valid only within the session in which they are generated.

5.1.1.2. Card Content Management

O.INSTALL

The TOE shall ensure that the installation of an application is safe.

In order to be safe, the installation process must satisfy the following requirements:

- The TOE must be able to undo all the installation steps when the installation fails or is cancelled, whatever the reasons.
- Installing an application must have no effect on the code and data of already installed applets. In particular, the installation of a new application or Executable File shall not hide or make inaccessible any other application or file already existing on the card.

The installation procedure should not be used to bypass the TSFs. It shall be a secure atomic operation, and free of harmful effects on the state of the other applets.

In particular, the set of privileges granted to an application shall be consistent with the intended meaning of each privilege and with the configuration of GP implemented by the platform.

The procedure of installing an application shall ensure the integrity and origin of the installation request, including the privileges granted to the application.

If the card has been set to the Closed Mode, it must reject any attempt of creating a new applet instance, even if this action is required by an externa user authenticated as the Card Administrator.

O.LOAD

The procedure of loading and installing an Executable Load File shall ensure the integrity and authenticity the file.

The TOE must check that each Executable Load File actually comes from the Card Administrator, who has previously checked that it is harmless for the other applications installed on the card. Moreover, for further security, the card shall check the DAP signature of the Verification Authority in charge of performing that checkings. This signature is attached to the Executable Load File.

If the card has been set to the Closed Mode, it must reject any attempt of loading an Executable File, even if this action is required by an externa user authenticated as the Card Administrator.

O.DELETION

The TOE shall ensure that both application and Executable File deletion are safe.

The deletion mechanism shall consider the following issues:

- Deletion of installed applets (or Executable Files) shall neither introduce security holes in the form of broken references to garbage collected code or data, nor alter the integrity or confidentiality of the remaining applets. The deletion procedure shall not be maliciously used to bypass the TSF.
- 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. The deletion of an Executable File shall make its code no longer available for execution.
- Power failure or other failures during the process shall be taken into account in the implementation so as to preserve the TSPs. This does not mandate the whole process to be atomic, but rather that it can be sliced into small and atomic deletion steps. For instance, an interrupted deletion may result in the loss of user data, as long as it does not violate the TSPs.
- It shall not be possible to delete the Executable Files corresponding to Java Card and GlobalPlatform's API.

Application note:

Logical deletion is only acceptable for those Executable Files stored in ROM. Deleted Executable Files stored in EEPROM shall be physically removed from the smart card memory.

5.1.1.3. Life Cycle Management

O.LIFE-CYCLE

The TOE shall enforce a well-defined life-cycle, keeping track of its current state, and controlling that the operations required by the users are consistent with the current life cycle state of the TOE.

The life cycle shall include a well-identified life cycle state after which the TOE is in a secure state and immediately operational, and prevent the execution of those functions that could become insecure. Examples of functions that would downgrade the security of the TOE in the operational state are testing and debugging functions or loading a Patch File after platform initialization. The life cycle shall also include a final state, where the TOE is definitely terminated.

O.IDENTIFICATION

The TOE must provide means to store initialization data and platform personalization data in its non-volatile memory.

The initialization data (or parts of them) are used for TOE identification and for tracing the actors that were involved in the previous phases of the TOE's life cycle.

Application note:

Initialization data includes the Card Production Life Cycle data specified in [VGP].

5.1.1.4. Cardholder Verification Method

O.GLOBAL-CVM

The TOE shall enable the applications to consistently manage a unique service for authenticating the Cardholder (CVM service).

The TOE shall restrict the modification of the security attributes of the CVM only to some privileged applications appointed by the Card Administrator. Only the Card Administrator may grant the CVM privilege to an applet.

O.CVM-BLOCK

No further Cardholder authentication attempts shall be possible once the maximal number of attempts has been reached, until a special action is performed by a privileged user.

5.1.1.5. Logical Protection

O.SID

The TOE shall uniquely identify every applet instance before granting it access to any security sensitive service.

The TOE shall only accept requests coming from application instances that have been correctly registered by the OPEN in the GlobalPlatform's registry. The security functions offered through the Java Card or GlobalPlatform's API shall always reject requests coming from applications that have not yet been registered.

Application note:

An application instance that is not yet registered cannot share objects with other application instances, nor set the PIN code of the CVM. As a consequence, those services are rejected when invoked from the `Applet.install` method before that method registers the applet instance.

O.ERROR-COUNTERS

The TOE must prevent the release of information through the analysis of responses to repetitive stimulations. This objective could also work through the detection of such attacks and the initiation of corrective actions to counter such attempts.

Detection involves journalizing how many times the same unsuccessful operation has been carried out, like the number of unsuccessful authentication attempts made by the Cardholder. Corrective actions involve temporarily or definitely locking some of the services that the card provides.

O.RECOVERY

The TOE shall check at the beginning of each session whether there still remained some unfinished tasks when the card was powered off. If there are unfinished tasks, the TOE shall either complete them (if possible), or abort the whole action that gave rise to them and roll back to a safe state.

If power is lost or if the smart card is withdrawn from the CAD while an atomic operation is in progress, the TOE must eventually complete the interrupted operation successfully, or recover to a consistent and secure state. Atomic operations are, for instance, loading new secret keys, downloading an Executable File, or installing or deleting an applet instance.

Application note:

In the [JCSPP], this objective is part of the objectives for the security environment. In this Security Target it has been included among the security objectives for the TOE because the scope of the TOE includes the Operating System, which implements atomic transactions.

O.LOCK

The TOE shall enable the applications to temporarily or definitely disable the services they provide, or even the services provided by the whole platform. The TOE shall restrict the use of such sensitive capabilities only to privileged applications appointed by the Card Administrator.

Entering into a state where the set of services is restricted is a countermeasure that the applets may use in response to a security violation. This counter-measure has to be restricted to appointed applications in order to prevent a possible denial of service attack.

5.1.2. Objectives for the Runtime Environment

This section introduces the security objectives that are relative to the runtime environment on which the applets are executed.

5.1.2.1. Execution of applets

The security objectives in this section concern the way in which the code of the applet instances is executed.

O.OPERATE

The TOE must ensure continued correct operation of its security functions.

The TOE must also return to a well-defined valid state before a service request in case of failure during its operation.

O.RESOURCES

The TOE shall control the availability of resources for the application instances.

The TOE must enforce quotas and limitations in order to prevent unauthorized denial of service or malfunction of the TSF. This concerns both execution (dynamic memory allocation) and installation (static memory allocation) of application instances and Executable Files.

O.FIREWALL

The TOE shall provide controlled sharing of data containers owned by Applet instances of different Executable Files, and between applet instances and the TSF.

The Platform shall ensure controlled sharing of class instances and arrays, and isolation of the data and the code of different Executable Files (that is, controlled execution contexts).

An applet instance shall neither read, write nor compare a piece of data belonging to an instance of another applet that is not in the same context, nor execute one of the methods of an instance of an applet in another context without its authorization.

O.REALLOCATION

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

O.SHRD_VAR_CONFID

The TOE shall ensure that any data container that is shared by all application instances is always cleaned after the execution of an application.

Examples of such shared containers are the APDU buffer, the byte array used for the invocation of the process method of the selected applet, or any public global variable exported by the Java Card API.

O.SHRD_VAR_INTEG

The TOE shall ensure that only the currently selected application instance may grant write access to a data memory area that is shared by all application instances, like the APDU buffer, the byte array used for the invocation of the process method of the selected applet, or any public global variable exported by the API.

Even though the memory area is shared by all applet instances, the TOE shall restrict the possibility of getting a reference to such memory area to the application that has been selected for execution. The selected applet instance may decide, at its own risk, to temporarily hand over the reference to other applet instances, but the TOE shall prevent those instances from storing the reference as part of their persistent states.

O.NATIVE

The *only* means that the JCVM shall provide for an applet instance to execute native code is the invocation of a method of the Java Card API.

Because the direct execution of arbitrary native code is beyond the control of the JCVM, it must be secured so as not to provide ways to bypass the TSFs. No distrusted native code may reside on the card. Loading of native code into the Platform is submitted to the same requirements.

O.VERIFICATION

The TOE shall enforce runtime verifications to ensure the adequacy of each bytecode operand to the intended semantics of that bytecode.

Runtime verifications shall include checking at least the following properties:

- bytecode instructions represent a legal set of instructions used on the Java Card platform;
- partial adequacy of bytecode operands to bytecode semantics;
- absence of operand stack overflow/underflow;
- monitoring of control flow confinement to the current package code (detection of control jumps outside the current Method Component);
- forged references to a class instance or an arrays are rejected;
- illegal offsets into a class instance, array, static field image or local variable are rejected;
- native method invocation is restricted to the set of reliable ones masked with the Embedded Software.

Application note:

This objective is close to the O.VERIFICATION objective included in [JCSPPD]. The verifications that jTOP's defensive virtual machine enforce are not as large as the ones provided by an on-card bytecode verifier, but they nevertheless ensure the main security properties expected from it.

O.OS-SUPPORT

The Operating System layer shall include low-level support to the TSF of the Java Card Runtime Environment.

The support that the TOE shall provide to the Java Card Runtime Environment includes:

- Appropriate use of the MMU for protecting TSF data and TSF code against disclosure or modification by the applet instances.
- Atomically updating a collection of persistent memory positions;
- Detecting segmentation faults for the blocks allocated by the JCVM.

5.1.2.2. Services offered to the applets

The security objectives of this section concern the services that the Runtime Environment offers to the applet instances through the Java Card API.

O.PIN-MNGT

The TOE shall provide a means to securely manage PIN objects.

Secure management of PIN objects includes:

- Atomic update of PIN code and of the try counter,
- No rollback of the number of unsuccessful authentication attempts,
- Encryption of the PIN value and no clear-PIN-reading function,
- Enhanced protection of the PIN's security attributes,
- Software countermeasures to make it difficult to observe PIN comparisons.

O.ALARM

The TOE shall provide appropriate feedback information upon detection of a potential security violation.

This concerns in particular the security exceptions thrown by the JCVM, or any other security-related event occurring during the execution of a TSF.

O.TRANSACTION

The TOE must provide a means to atomically execute a sequence of operations.

To atomically execute a sequence of operations means that either all the operations are completely executed or the Runtime Environment behaves as if none of them would be executed. Applet instances may request the platform to perform a sequence of modifications atomically through the Java Card API

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.

Ciphering includes the following cryptographic operations: data encryption and decryption, electronic signature generation and verification, computation and update of a hash value, computation and update of a checksum and random number generation. These operations shall include mechanisms to resist to the SPA/DPA/DFA/EMA techniques that are part of the state of the art. The cryptographic services shall also ensure that only keys that are consistent with the specified algorithm are used, and prevent any use of the services before it has been correctly initialized.

O.KEY-MNGT

The TOE shall provide means to securely manage cryptographic keys. This concerns the correct generation, distribution, access and destruction of cryptographic keys, including the following points:

- Keys shall be generated in accordance with specified cryptographic key generation algorithms and specified cryptographic key sizes,
- Keys shall be distributed in accordance with specified cryptographic key distribution methods. In particular, keys must be loaded through a secure channel ensuring key origin, authenticity and confidentiality.
- Keys shall be initialized before being used. In particular, the ISD shall verify that all the components of a loaded DES key have been received and checked before using the key.
- Keys shall be stored in secure containers that prevent their disclosure by direct observation of the smart card memory (memrory dumping)
- Keys shall be destroyed in accordance with specified cryptographic key destruction methods. These methods shall include the possibility of limiting the lifetime of the key value to the current session.

O.OBJ-DELETION

The TOE shall ensure that object deletion shall not introduce dangling pointers, and that garbage-collected information is always cleared.

Object de-allocation 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. This process should not be maliciously used to circumvent the TSF, like the Firewall. Erasure, if deemed successful, shall ensure that the deleted class instance is no longer accessible and that its contents have been cleared.

5.1.3. Objectives for the Integrated Circuit

The following objectives for the TOE come from [ICST]. The objective O.IDENTIFICATION of that Security Target is covered by the abovementioned objective of the same name. The objectives O.Leak-Inherent and O.Leak-Forced have been gather together into a single goal O.PROT-INF-LEAK. Similarly, the objectives O.Phys-Probing and O.Phys-Manipulation have been collapsed into a single goal O.PROT-PHYS-TAMPER. The O.ABUSE-FUNC objective stated in [ICST] is not included in this section, as it is already covered by the O.LIFE-CYCLE objective introduced above.

An extra security objective OE.IC.SUPPORT has been added to the ones introduced in [ICST]. This objective concerns the support to the Runtime Environment that is expected from the chip. It corresponds to the goals introduced for the IC in the OE.SCP.SUPPORT objective stated in [JCSPP] and includes those additional features introduced in the O.Add-Functions objective in [ICST] that are actually used by the Embedded Software.

When appropriate, the term "TOE" used in [ICST] has been replaced in the statement of the objectives by the more precise term "IC", as the TOE of this Security Target also includes the Embedded Software.

O.IC-SUPPORT

The IC must provide the following specific security functionality to the Embedded Software:

- Area based Memory Access Control
- Triple Data Encryption Standard (3DES)

The IC shall be designed in accordance with the well-defined set of policies and standards specified in [ICST], 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 is especially important for the management (storage and operation) of cryptographic keys. It shall also support the TSF of the Runtime Environment providing physical isolation between memory blocks.

Application note:

This security objective is called O.Add-Functions in [ICST].

O.PROT-INF-LEAK

The IC must provide protection against disclosure of confidential TSF data stored and/or processed in the smart card's chip.

Protection must cover the disclosure of TSF data by:

- measurement and analysis of the shape and amplitude of signals or the time between events found by measuring signals on the electromagnetic field, power consumption, clock, or I/O lines, and:
- forcing a malfunction of the TOE and/or a physical manipulation of the TOE.

Application note:

This objective pertains to measurements with subsequent complex signal processing due to either normal operation of the TOE or operations enforced by an attacker.

O.PROT-PHYS-TAMPER

The IC must ensure the confidentiality and integrity of the User Data, the TSF Data, and the smart card's chip Embedded Software.

This includes protection against attacks with high attack potential by means of:

- measuring through galvanic contacts which is direct physical probing on the chips surface except on pads being bonded (using standard tools for measuring voltage and current); or
- measuring not using galvanic contacts but other types of physical interaction between charges (using tools used in solid-state physics research and IC failure analysis)
- manipulation of the hardware and its security features; as well as
- controlled manipulation of memory contents (User Data, TSF Data).

All these attacks require prior reverse-engineering to understand the design and its properties and functions.

O.PROT-MALFUNCTION

The IC must ensure its correct operation.

The IC must refuse to operate outside normal conditions where reliability and secure operation have not been proven or tested. This is needed to prevent induced errors. The environmental conditions may include external energy (esp. electromagnetic) fields, voltage (on any contacts), clock frequency, or temperature.

Application note:

A malfunction of the TOE may also be caused using a direct interaction with elements on the chip surface. This is considered to be a manipulation (refer to the objective O.PROT-PHYS-TAMPER) provided that detailed knowledge about the TOE´s internals is available.

O.RND

The IC must contribute to ensure that random numbers shall not be predictable and shall have sufficient entropy.

5.2. Security objectives for the environment

This section introduces the security objectives that the environment of the TOE shall meet. Security objectives on the environment are related to assumptions or organizational security policies. In some cases they are just a repetition of the assumptions that the security objectives have to uphold.

The scope of this Security Target enlarges that of [JCSPP] to include the Card Manager, the Operating System and the IC. Consequently, some of the security objectives for the environment introduced in [JCSPP] have been either transformed into security objectives for the TOE or replaced by more specific goals taken from [ICST], or by new specific objectives stated for the Card Manager. This is the case of the following security objectives of the environment in [JCSPP]:

- OE.RECOVERY has been transformed into the security objective O.RECOVERY;
- OE.SUPPORT is partially covered by O.ABUSE-FUNC, O.PROT-MALFUNCTION and O.RND. Those properties that are not covered by the objectives in [ICST] have given raise to the security objective O.OS-SUPPORT and O.IC-SUPPORT.
- OE.SCP.IC has been replaced by the more specific objectives O.PROT-PHYS-TAMPER and O.PROT-INF-LEAK
- OE.CARD-MANAGEMENT is discharged by the objectives regarding card management that were stated in the previous section.

This Security Target also enlarges the scope of [ICST] so as to include the Embedded Software. As a consequence, those security objectives for the IC environment that refer to the correct use of the hardware by the Platform Developer are not pertinent for this Security Target, as they are discharged (represented) either by the assurance measures or by the objectives for the TOE. This is the case of the following security objectives introduced in [ICST]:

- OE.Plat-Appl, which concerns the correct use of the hardware, is mainly covered by the ADV and ALC assurance measures.
- OE.Resp-Appl, which concerns the correct treatment of cryptographic keys by the Embedded Software, is covered by O.KEY-MNGT, O.INFO-ORIGIN, O.INFO-INTEGRITY and O.INFO-CONFIDENTIALITY.

Other security objectives not included in the [JCSPP] have been introduced in order to support the assumptions made in this Security Target. This is the case of OE.MANUFACTURING. This latter objective covers the OE.Process-TOE and OE.Process-Card security objectives for the environment defined in [ICST]. Those objectives concern the protection of the IC all along the card manufacturing phase, which includes all the roles involved in the Platform Developer phase of the TOE: Platform Developer, Application Provider, IC Manufacturer, Card Manufacturer and Card Enabler.

OE.VERIFICATION

The Card Administrator transmits an Executable Load File to the card only if he/she applies the security recommendations in [USR], has successfully passed the Bytecode Verification process and has not been modified afterwards.

Bytecode verification shall include:

- well-formedness of the CAP file structure and verification of the typing constraints on its bytecodes,
- binary compatibility with installed Executable Files and the assurance that the export files used to check the Executable Load File match the CAP files that will be present on the card when loading occurs.

OE.NATIVE

The Platform Developper shall ensure that all pre-issuance native applications masked with the code of the platform enforce the security policies and objectives described in this Security Target.

In particular, native applications that handle Java Card objects must respect the Java Card Firewall policy.

Application note:

In [JCSPP], this security objective also requires that parts of the API that are implemented as native methods enforce the security policies defined for the platform. That part of the objective has been discharged in this Security Target, as the native libraries of the Operating System that support API implementation are also in the scope of this TOE.

OE.APPLET

The applets downloaded during the post-issuance phase must not contain native methods.

OE.SECRETS

The attacker shall not be able to obtain neither the private key for generating DAP signatures nor any of the PIN codes or secret keys stored in the card from the TOE IT or non-IT environment.

OE.KEY-LENGTH

The Verification Authority shall only validate those applets that respect the constraints regarding cryptographic key lengths provided in the Security Functional Requirements of this document.

Application note:

The precise list of cryptographic algorithms defined in the Java Card API that fall into the scope of evaluation is summarized in the document [PROFILE].

OE.NO-RMI-APPLETS

The Card Administrator shall not load applets based on the Remote Method Invocation (RMI) mechanism.

OE.MANUFACTURING

The Platform Manufacturer shall ensure the quality and integrity of the manufacturing process as well as the definite disabling of all test and debug mechanisms from the product once the smart card enters the platform personalization phase.

The Platform Manufacturer and the Card Enabler must use only proper secret keys (chosen from a large key space) as input for the cryptographic function of the TOE.

The different actors embodying the role of the Platform Manufacturer (Application Providers, the Platform Developer, the IC Manufacturer, Card Manufacturer) and Card Enabler shall apply security procedures to maintain confidentiality and integrity of the TOE and of its manufacturing and test data up to delivery to the end-user (to prevent any possible copy, modification, retention, theft or unauthorized use). In particular, the TOE must be protected appropriately when it is delivered from one actor to the other.

6 IT security requirements

6.1. TOE security functional requirements

This section describes the requirements imposed on the TOE in order to achieve the security objectives that were laid down for it in the previous chapter. All the requirements identified in this section are instances of those stated in [CC2], except for the instance of the FCS_RND family, which has been taken from [SSVG]. The definition of the FCS_RND family is recalled as an Appendix of this Security Target.

Requirements are classified into different categories, according to the component of the platform to which they are more closely related: Card Management, Runtime Environment or Smart Card Platform.

The minimum SOF level for the TOE security functional requirements is SOF-High.

6.1.1. Card Management

This section describes the security functional requirements imposed on card management operations issued from the *CMGR* group of requirements introduced in [JCSPP]. Opposite to [JCSPP], and according to the scope defined for the TOE, those requirements are imposed in this Security Target on the TOE instead of on its IT Environment.

Card Management requirements are gathered into the following categories:

- Issuer Security Domain
- Secure Channels
- Card Content Management
- Global Cardholder Verification Method
- Patch Management

Each category contains the security requirements concerning the implementation of a specific feature of GlobalPlatform.

6.1.1.1. Issuer Security Domain

This section introduces an access control policy containing the requirements that are specific to the card management operations that GlobalPlatform provides: enforcing the life cycle of the card and its applets, loading and installing new applets, removing existing ones, loading new cryptographic keys, and setting and retrieving information about the card's history. This policy corresponds to the (unassigned) *Card Content Management* access control policy introduced in [JCSPP]. The policy has been renamed here as the *Issuer Security Domain* access control policy, because it enforces the policy that GlobalPlatform's specifications assigns to that distinguished application. That policy does not only concern card content management, but also other aspects of card management activities, like controlling the life cycle of the card and the applets. Accordingly, the suffix «ISD» replaces the suffix «CMGR» used in [JCSPP] for naming the components of this family.

FDP_ACC.1-ISD Subset access control

FDP_ACC.1.1-ISD The TSF shall enforce the **Issuer Security Domain (ISD) access control policy** on **the following list of subjects, objects and operations:**

- **The subjects controlled by the policy are the applications instances installed on the card. Among those applications there is a distinguished one, called the Issuer Security Domain.**
- **The objects controlled by the policy are:**
  - **The Executable Files loaded on the platform**
  - **The Executable Modules defined in the Executable Files**
  - **The application instances created on the card**
  - **The keys handled by the security domain**
  - **The personalization data for both the card and the applications**
- **The operations controlled by the policy are GlobalPlatform's APDU commands and API methods**.

Application note:

GlobalPlatform specifies the following APDU commands:

- DELETE: deleting an Application Instance or an Executable File
- EXTERNAL AUTHENTICATE: authenticating the host and determining the security level of the secure channel
- GET DATA: retrieving card administration information
- GET STATUS: retrieving loaded keys, installed Executable Files and Application instances, and their current life cycles
- INITIALIZE UPDATE: requesting the opening of a secure channel
- INSTALL[for install and make selectable]: installing a new application instance
- INSTALL[for load]: installing a new Executable Load File
- INSTALL[for registry update]: preventing loading any further Executable File on the card
- SET PACKAGE PROPERTIES: restricting card content actions on a given Executable File
- LOAD: loading of a piece of a Executable Load File
- PUT KEY: loading or replacing a key set
- SELECT: selecting of an application instance for execution
- SET STATUS: modifying the life cycle state of either the card or an application
- STORE DATA: loading of personalization data, including ISD static keys The ISD access control policy also controls the use of the following methods of the GlobalPlatform API:
- *GPSystem.setCardContentState(state)*: modification of the life cycle state of an application instance
- *GPSystem.terminateCard()*: definitely disabling the services of the smart card
- *GPSystem.lockCard()*: temporarily blocking the services of the smart card
- *CVM.update()*
- *CVM.blockState()*
- *CVM.resetAndUnblockState()*
- *CVM.setTryLimit()*

FDP_ACF.1-ISD Security attribute based access control

FDP_ACF.1.1-ISD The TSF shall enforce the **Issuer Security Domain access control policy (ISD)** to objects based on the following:

All application instances, including the ISD, have the following security attributes:

- **The *Closed Mode* attribute specifies whether the TOE has been configured to prevent the loading of additional Executable Files.**
- **The *Default Selected* attribute specifies whether the applet instance is the one that should be executed when no application has been explicitly selected.**
- **The *Application State* attribute specifies the current life cycle state of the application instance, which may be either SELECTABLE, APPLICATION_SPECIFIC,LOCKED.**
- **The *Card Lock* attribute specifies whether the applet is allowed to temporary lock the services of the smart card**
- **The *Card Termination* attribute specifies whether the applet is allowed to definitely disable the services of the smart card.**
- **The *CVM* attribute specifies whether the applet is allowed to modify the try limit and the PIN code of the global CVM service. In addition to the ones above, the ISD has the following extra security attributes:**
- **The *CardState* attribute, is the current state in the life cycle of the card, which may be either OP_READY, INITIALIZED, SECURED, CARD_LOCKED, or TERMINATED.**
- **The *Registered Applications* attribute specifies the Executable Files and application instances that have been installed on the card so far and their dependencies.**

The packages under the control of this security policy have the following attributes:

- **The *Package Properties* attribute specify whether the following restrictions apply to the applet classes that the package declares: selection of applet instances in contact-based mode is disabled.**
  - **selection of applet instances in contactless mode is disabled.**
  - **installation of additionnal instances of those classes is disabled.**
  - **installation of additional instances is disabled, provided that one single instance has been created at the moment of the installation request.**
  - **deletion of any instance of those classes is disabled.**.

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

- **Rule ISD-1: A card administration request may be accepted only if the APDU command specifying the request is well-formed according to [VGP].**

- **Rule ISD-2: A card administration request other than requesting card management data may be accepted only if the *Card State* is not TERMINATED.**
- **Rule ISD-3: The selection of an applet instance different from the ISD may be accepted only if the properties of the package declaring the applet enable selection through the interface used to send the command, and the *Applet State* is not LOCKED.**
- **Rule ISD-4: The update of the life cycle state of an application instance is accepted only if the new state is consistent with its current life cycle state according to GlobalPlatform's life cycle rules (either coming from an APDU command or from an application instance through the GP API).**
- **Rule ISD-5: A request for installing an Executable Load File may be accepted only if the card has not been set to the Closed Mode, there is enough resources for loading the Executable File, and no Executable File on the card has been already registered with the specified AID.**
- **Rule ISD-6: A Executable Load File block may be loaded only if the card has not been set to the Closed Mode, all its previous blocks have been received in order, and there are sufficient resources for storing the new one.**
- **Rule ISD-7: A new applet instance may be created only if the card has not been set to the Closed Mode, the Package Properties enables applet instantiation and (if there is already an instance for that applet) multiple applet instances, the AID specified for the applet instance is not already used for another applet or Executable File installed on the card, and the privileges specified for it are consistent with the configuration of GlobalPlatform specified in [VGP].**
- **Rule ISD-8: An Executable File may be deleted from the smart card only if its Package Properties enable deletion, it is not reachable from other Executable Files or application instances on the card, it is not is not necessary for the working of the platform, like for instance, the Java Card or GlobalPlatform APIs.**
- **Rule ISD-9: An applet instance may be deleted from the card only if the instance is not the ISD, it is not currently active on a logical channel, and none of the Java Card objects it has allocated is reachable from other Executable Files or Application instances installed on the card.**
- **Rule ISD-10: An applet instance may lock the card only if it has the *Card Lock* privilege.**
- **Rule ISD-11: An applet instance may terminate the card only if it has the *Card Termination* privilege.**
- **Rule ISD-12: An applet instance may unlock the CVM service or modify the CVM try limit or PIN code only if it has the *CVM* privilege.**
- **Rule ISD-13: A request involving the use of any of the ISD keys is accepted only if the concerned keys are integer with respect to their associated checksum values.**.

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

FDP_ACF.1.4-ISD The TSF shall explicitly deny access of subjects to objects based on the **following rule: when at least one of the conditions listed in the element FDP_ACF.1.2 of this component does not hold**.

FMT_MSA.1-ISD.1 Management of security attributes

FMT_MSA.1.1-ISD.1 The TSF shall enforce the **Issuer Security Domain access control policy (ISD)** to restrict the ability to **modify** the security attributes ***Application State*** **of an application instance** to **the Issuer Security Domain and the application instance itself**.

FMT_MSA.1-ISD.2 Management of security attributes

FMT_MSA.1.1-ISD.2 The TSF shall enforce the **Issuer Security Domain access control policy (ISD)** to restrict the ability to **modify** the security attributes ***Default Selected Application*** to **the Issuer Security Domain**.

FMT_MSA.1-ISD.3 Management of security attributes

FMT_MSA.1.1-ISD.3 The TSF shall enforce the **Issuer Security Domain (ISD) access control policies** to restrict the ability to **modify** the security attributes ***Card State*** to **the Issuer Security Domain, the Card Lock Privileged Applets and the Card Termination Privileged Applets**.

FMT_MSA.1-ISD.4 Management of security attributes

FMT_MSA.1.1-ISD.4 The TSF shall enforce the **Issuer Security Domain (ISD) access control policies** to restrict the ability to **modify** the security attributes ***Closed Mode, Package Properties*** to **the Issuer Security Domain**.

Application note:

Depending on how the card was configured during its initialization, the Issuer Security Domain may either close it automatically when the card reaches GlobalPlatform's SECURED state, or upon the reception of a some APDU commands from the Card Administrator.

Package Properties may be modified using the SET PACKAGE PROPERTIES command.

FMT_MSA.3-ISD Static attribute initialisation

FMT_MSA.3.1-ISD The TSF shall enforce the **Issuer Security Domain access control policy (ISD)** to provide **the belowmentioned** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2-ISD The TSF shall allow the **Card Administrator** to specify alternative initial values to override the default values when an object or information is created.

Global refinement:

- The *Default Selected* application shall be the ISD.
- When the TOE enters the life cycle phases under the scope of this Security Target, the *Card State* shall be at least INITIALIZED.
- The initial value of the *Application State* of an applet instance shall be SELECTABLE.
- The initial *Package Properties* shall enable all card content management operations on the package.

Application note:

The Card Administrator may assign the Default Select privilege to another application instance. The default value of the other security attributes of the ISD policy cannot be modified.

FMT_SMF.1-ISD Specification of management functions

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

- o **Setting the card to the Closed Mode, in which no further Executable Files can be loaded.**
- o **Restricting the properties associated to a given package. - Registering a new Executable File or application instance in the GlobalPlatform's registry.**
- o **Removing the specified entries from the GlobalPlatform registry when a DELETE command is received.**
- o **Unsetting it as the Default Select application and set this privilege to a new application instance.**
- o **Granting the privileges that the Card Administrator specifies when an new application instance is installed.**
- o **Moving the life cycle state of either an application instance or the whole card according to the life-cycle rules specified in [VGP]**.

FMT_SMR.1-ISD Security roles

FMT_SMR.1.1-ISD The TSF shall maintain the roles **Issuer Security Domain**.

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

FMT_SMR.1-PRV Security roles

FMT_SMR.1.1-PRV The TSF shall maintain the roles **Default Select Applet, Card Lock Privileged Applet, Card Termination Privileged Applet and CVM Privileged Applet**.

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

6.1.1.2. Secure Channels

This section introduces the information flow control policy controlling the exchange of security sensitive information between the card and the card host. It also introduces additional requirements concerning the operations that can be required by the card host before being identified and authenticated by the card.

The requirements introduced in this section correspond to the *Car* group of requirements of [JCSPP]. The goal of that group of requirements is to prevent the installation of an Executable File that has not successfully passed the bytecode verification, or that has been modified after having been verified. As the Card Administrator always verifies the bytecode loaded on the card, and the card provides a secure channel that prevents modifications, those requirements ensures that only correct bytecode can be loaded on the card.

The Carrier Group introduces security requirements concerning the generation of evidence of origin and the integrity of the Load Files downloaded on the card. In [JCSPP], the integrity requirement depends on an information flow control policy called PACKAGE LOADING policy describing the rule to be enforced in order to prevent the modification of the Load File while it is in transit from the CAD to the card. In this Security Target, the PACKAGE LOADING policy is instantiated with the Secure Channel Protocol one, which prevents the insertion or modification of the Load File piece. Accordingly, the suffix «SCP» replaces the suffix «CM» used in [JCSPP] for naming the components of this family

Secure Channel Actions

FTP_ITC.1-SC Inter-TSF trusted channel

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

FTP_ITC.1.2-SC The TSF shall permit **the remote trusted IT product** to initiate communication via the trusted channel.

FTP_ITC.1.3-SC The TSF shall initiate communication via the trusted channel for **loading or deleting an Executable File on the card, installing or removing an application instance, loading or replacing a key set, personalizing the platform, personalizing an application, and changing the life cycle state of an application or the whole card**.

Application note:

The remote IT product mentioned in the second element of this component is the CAD placed in the Card Administrator secured environment.

This Security Target refines the requirement FTP-ITC.1/CM in [JCSPP] by requesting the use of a secure channel for other security-sensitive card management operations than the loading of a new Executable File.

FCO_NRO.2-SC Enforced proof of origin

FCO_NRO.2.1-SC The TSF shall enforce the generation of evidence of origin for transmitted **Executable Load Files** at all times.

FCO_NRO.2.2-SC The TSF shall be able to relate the **identity** of the originator of the information, and the **Executable Load Files** of the information to which the evidence applies.

FCO_NRO.2.3-SC The TSF shall provide a capability to verify the evidence of origin of information to **the Card Issuer** given **immediate verification**.

Application note:

The origin of the Executable File is verified using GlobalPlatform's protocol SCP02. In the mutual authentication step of that protocol, the Card Administrator and the Issuer Security Domain agree on a session key based on a shared secret. If the Executable Load File is loaded in a secure physical environment, this authentication is sufficient to prove the origin of the file. If the Executable Load File is transmitted through an unsecure transmission media, the Card Issuer may be related to the whole Executable Load File by the verification of a (chained) MAC that the Card Administrator attaches to each file block sent to the card. Once the session is finished, the session key used to verify the MAC is cleared, so verification of evidence can only be made inside the downloading session (immediate verification).

If the card has been configured to do so, in addition to the verification of the origin of each Executable Load File block, the Issuer Security Domain also verifies the DAP signature on the whole Executable File attesting that the Verification Authority has validated it. This signature is verified before installing the Executable File but it is not recorded for future verification.

In [JCSPP], the list of third parties that may verify the origin of an Executable Load File is instantiated with *recipient*. An application note in [JCSPP] explains that the intended recipient is a Verification Authority responsible for bytecode verification. In this configuration of GlobalPlatform, this role is played by the Card Issuer. Consequently, the term "recipient" in [JCSPP] is replaced in this Security Target by "the Card Administrator".

Identification and Authentication

This section introduces the requirements concerning the actions that a subject may perform before opening a Secure Channel with the card.

FIA_UID.1-SC Timing of identification

FIA_UID.1.1-SC The TSF shall allow **the mediated actions listed below** on behalf of the user to be performed before the user is identified.

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

Global refinement:

For those card management operations performed through APDU commands, the following mediated actions are allowed before identification:

- selecting an application on the card;
- requesting data that identifies the card or the Card Issuer, provided that this feature has not been disabled during the Platform Initialization Phase;
- initializing a secure communication channel with the card.

Application note:

The actions that can be performed before identification in this security requirement are specific to the opening of a secure communication channel with the CAD. Other mediated actions regarding other identification process are listed in the component FIA_UID.2-AID.

FIA_UAU.1-SC Timing of authentication

FIA_UAU.1.1-SC The TSF shall allow **the TSF mediated actions listed in FIA_UID.1-SC** on behalf of the user to be performed before the user is authenticated.

FIA_UAU.1.2-SC The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

FIA_UAU.4-SC Single-use authentication mechanisms

FIA_UAU.4.1-SC The TSF shall prevent reuse of authentication data related to **the authentication mechanism used to open a secure communication channel with the card**.

SCP Information Flow Security Policy

This section introduces the requirements to be imposed on the Secure Channel linking the card and the card host when the channel is explicitly started. Those requirements come from the Secure Channel Protocols introduced in [GPCS].

FDP_IFC.2-SCP Complete information flow control

FDP_IFC.2.1-SCP The TSF shall enforce the **Secure Channel Protocol information flow control policy (SCP)** on **the subjects and information listed below** and all operations that cause that information to flow to and from subjects covered by the SFP.

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

Global refinement:

- The subjects covered by this policy are those involved in the exchange of messages between the card and the CAD through a potentially unsafe communication channel:
  - An off-card subject that represents the Card Administrator (S.BCV).
  - Any application with the Security Domain privilege (S.CRD).
  - Any other subject that may potentially insert, delete, modify, or permute the messages exchanged between the two former subjects (S.SPY).
- The information controlled by this policy is the one contained in the APDU commands sent to the card and their associated responses returned to the CAD. This includes Executable Files, file and application names, application privileges, application and card life-cycle states, smart card and Card Issuer identification data, the Issuer Security Domain keys, personalization data, and the requests to and the responses from the services offered by the embedded applications

Application note:

The operations that make information to flow between the subjects are to send a message through and receive a message from the communication channel linking the card to the outside world. Such messages are the GlobalPlatform commands and their associated responses, which are schematically characterized as follows:

- SEND(M): A subject sends a message M through the communication channel.
- RECEIVE(M): A subject receives a message M from the communication channel.

The configuration of GlobalPlatform that the TOE features is such that only the Issuer Security Domain has the Security Domain privilege.

The subjects controlled by this security policy are the same as in [JCSPP], but they have been renamed according to GlobalPlatform's terminology:

- The off-card entity responsible for the bytecode verification of the Executable Load File, which is named S.BCV in [JCSPP], is called Card Administrator in this Security Target.
- The on-card entity responsible for the downloading of the Load File, which is called S.CRD in [JCSPP], is called the Issuer Security Domain in this Security Target.
- The attacker, called S.SPY in [JCSPP], is also called Spy in this Security Target.

FDP_IFF.1-SCP Simple security attributes

FDP_IFF.1.1-SCP The TSF shall enforce the **Secure Channel Protocol information flow control policy (SCP)** based on the following types of subject and information security attributes:

- o **The messages exchanged between the on-card and the off-card subjects have a single security attribute, namely, the *MAC* ensuring the integrity and the origin of the message**
- o **The on-card and the off-card subjects have the following security attributes:**
  - ▪ **The *Challenge* is a random number generated by the subject in order to identify the current session.**
  - ▪ **The *Cryptogram* is a secret relative to the current smart card session that serves to authenticate the on- and off-card subjects (but not the spy). The cryptogram is derived from the challenges of both the card and the terminal.**
  - ▪ **The *Key Set* is a collection of key triplets (called key set) used to encrypt the Derivation Data in order to generate the session keys. Each key set is composed of a Secure Channel Encryption Key (S-ENC), a Command Message Authentication Code Key (C-MAC) and a Data Encryption Key (DEK).**
  - ▪ **The *Static Keys* is a collection of key sets, each one identified by a key version number.**
  - ▪ **The *Session Keys* is a set of keys used to verify the origin and integrity of the received message, and to decrypt their contents. This set is made of the following keys:**
    - **Command Message Authentication Code Key (C-MAC session key);**
    - **Encryption Key (S-ENC session key);**
    - **Data Encryption Key (DEK session key).**
  - ▪ **The *Sequence Counter* is a counter attached to each Key Set, which is used to derive the session keys.**
  - ▪ **The *Initial Chaining Vector (ICV)* is a value used to compute the MAC value of a message, which relates it to the previous messages of the current session.**
- o **In addition to the abovementioned ones, the ISD have an extra attribute, namely, the *Command Security Level* defined for the**

messages that the card receives through the secure channel. The possible security levels are: NO-SEC (clear text), C-AUTHENTICATED (authentication of the command's issuer), C-MAC (authentication of the issuer and integrity of the command), C-DEC (authentication of the issuer, integrity and confidentiality of the command).

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

- **The ISD may process a RECEIVE(INITIALIZE-UPDATE) operation only if the key set specified in the command exist among the static keys of the ISD.**
- **The ISD may process a RECEIVE(EXTERNAL-AUTHENTICATE) operation if the following conditions hold:**
  - **The cryptogram received from the off-card subject is equal to the cryptogram computed by the Security Domain.**
  - **The MAC attached to the message has been generated from the C-MAC session key and the current value of the ICV.**
- **The ISD may process a RECEIVE (GET-DATA) operation if the following condition holds:**
  - **If the command security level is at least C-MAC, the MAC attached to the message has been generated from the command using the C-MAC session key and the current value of the ICV.**
- **The ISD may process a RECEIVE (M) operation for any other command M different from the ones cited in the rules above if the following conditions hold:**
  - **The current security level is at least AUTHENTICATED.**
  - **If the command security level is at least C-MAC, the MAC attached to the message has been generated from the clear-text command using the C-MAC session key and the current value of the ICV.**.

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

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

FDP_IFF.1.5-SCP The TSF shall explicitly authorise an information flow based on the following rules:

- **A Security Domain may process a RECEIVE(GET DATA) operation at the security level *NO-SEC*.**
- **A Security Domain may always process a RECEIVE(SELECT) operation**.

FDP_IFF.1.6-SCP The TSF shall explicitly deny an information flow based on the following rules: **when none of the conditions listed in the element FDP_IFF.1.5 of this component hold and at least one of those listed in the element FDP_IFF.1.2 does not hold**.

Application note:

The flow of messages in the rules of the policy is described from the card's side: when the operation is SEND, the subject sending the message is the on-card one, when the operation is RECEIVE, the subject receiving the message is the on-card one.

FMT_MSA.1-SCP-OFFCARD Management of security attributes

FMT_MSA.1.1-SCP-OFFCARD The TSF shall enforce the **Secure Channel Protocol (SCP) information flow control policy** to restrict the ability to **modify** the security attributes ***Key Set*, *Static Keys*, *Command Security Level* of a Security Domain** to **the Card Administrator**.

Application note:

This requirement specifies the security attributes that can be modified by the off-card users of which the Security Domain is the on-card representative. The value of those attributes is specified in the parameters of the commands that the off-card user sends to the Security Domain. The Key Set and the CAD Challenge are specified in the INITIALIZE-UPDATE command requesting the opening of a Secure Channel. The value of the Static Keys of the Security Domain is modified when the off-card user sends a PUT KEY command to the Security Domain. The Command Security Level is specified by the EXTERNAL-AUTHENTICATE command during the initialization of the Secure Channel. All those commands are actually processed only when the authentication step of the off-card user has been completed.

FMT_MSA.1-SCP-ONCARD Management of security attributes

FMT_MSA.1.1-SCP-ONCARD The TSF shall enforce the **Secure Channel Protocol (SCP) information flow control policy** to restrict the ability to **modify** the security attributes ***Session Keys*, *Sequence Counter* and *ICV*** to **the Issuer Security Domain**.

Application note:

In the SCP02 protocol, the ISD increments the *Sequence Counter* associated to the key set used to open the Secure Channel. The ISD updates the *ICV* each time it successfully processes the cryptographic protections of an APDU command. All those attributes are reinitialized each time the off-card subject request the opening of a Secure Channel.

FMT_MSA.1-SCP-BOTH Management of security attributes

FMT_MSA.1.1-SCP-BOTH The TSF shall enforce the **Secure Channel Protocol (SCP) information flow control policy** to restrict the ability to **modify** the security attributes ***Cryptogram* of a Security Domain** to **the Security Domain itself and its off-card counterpart (Card Administrator)**.

Application note:

The *Cryptogram* used by each subject to authenticate the other party is determined from the challenges exchanged by both subjects during the initialization of the Secure Channel.

FMT_SMF.1-SCP Specification of management functions

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

- **Generating a new card challenge during the set up of a Secure Channel.**
- **Generating the session keys for the Secure Channel from the specified static key set and its associated Sequence Counter.**
- **Generating the card cryptogram from the host and card challenges and the session keys.**
- **Increasing by one the Sequence Counter associated to the specified Key Set upon successful opening a Secure Channel.**
- **Setting the security level of the Secure Channel as the authenticated Card Administrator had specified during its set up.**
- **Updating the current value of the ICV upon reception of a new message through the Secure Channel.**
- **On request of the Card Administrator, loading or replacing the static keys that the associated Security Domain uses to open a Secure Channel**.

FMT_MSA.3-SCP Static attribute initialisation

FMT_MSA.3.1-SCP The TSF shall enforce the **Secure Channel Protocol (SCP) information flow control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2-SCP The TSF shall allow the **Card Administrator** to specify alternative initial values to override the default values when an object or information is created.

Global refinement:

The Card Administrator may specify alternative default values for the *Static Keys* attribute of the SCP policy.

Application note:

In [JCSPP], no role is authorized to provide alternative default values for the security attributes of the information flow control policy of the *Carrier* group. Such assignment seems unnecessary: as the protection profile does not specify which the security attributes of the policy are, stating that they cannot have alternative values seems to be an over-specification. For instance, such assignment is in conflict with GlobalPlatform's specifications. For this reason, this Security Target does not follow the Java Card Protection Profile on this point, and allows the off-card counterpart of a Security Domain to specify alternative default values for the static keys associated to a Secure Channel.

FMT_SMR.1-CA Security roles

FMT_SMR.1.1-CA The TSF shall maintain the roles **Card Administrator**.

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

Application note:

A Security Domain is just the on-card counterpart of the Card Issuer. It is introduced as a separate role in order to distinguish an application acting inside the smart card in behalf of the Card Issuer from the Card Administrator.

FIA_AFL.1-SC Authentication failure handling

FIA_AFL.1.1-SC The TSF shall detect when **one** unsuccessful authentication attempts occur related to **the authentication of the origin of a card management command**.

FIA_AFL.1.2-SC When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall **close the Secure Channel with the external user**.

Cryptography Operations

This section introduces the cryptography requirements concerning the verification of the origin, integrity and confidentiality of the card management commands received through a Secure Channel.

FCS_COP.1-SCP02/CBC Cryptographic operation

FCS_COP.1.1-SCP02/CBC The TSF shall perform **session key derivation and data field decryption of the messages exchanged through GlobalPlatform's Secure Channels** in accordance with a specified cryptographic algorithm **Triple DES in CBC mode** and cryptographic key sizes **of 112 bits** that meet the following: **FIPS PUB 46-3, ANSI X9.52 and ISO 10116**.

Application note:

Appendixes B, D and E of [GPCS] specify the padding and ICV to be used for each of the abovementioned cryptographic operations in GlobalPlatform 2.1.1.

FCS_COP.1-SCP02/ECB Cryptographic operation

FCS_COP.1.1-SCP02/ECB The TSF shall perform **key encryption/decryption and DES key's check value generation** in accordance with a specified cryptographic algorithm

(Triple DES in ECB mode) and cryptographic key sizes **of 112 bits** that meet the following: **FIPS PUB 46-3, ANSI X9.52 and ISO 10116**.

FCS_COP.1-SCP/FULL Cryptographic operation

FCS_COP.1.1-SCP/FULL The TSF shall perform **authentication cryptogram generation** in accordance with a specified cryptographic algorithm **(full triple DES)** and cryptographic key sizes **(16 bytes)** that meet the following: **FIPS PUB 46-3, ISO 9797-1**.

Application note:

Full triple DES is known is defined in [ISO 9797-1] as MAC Algorithm 1 without output transformation 1, without truncation, and with DES taking the place of the block cipher.

Appendixes B, D and E of [GPCS] specify the padding and ICV to be used for each of the abovementioned cryptographic operations in GlobalPlatform 2.1.1.

FCS_COP.1-SCP02/ICV Cryptographic operation

FCS_COP.1.1-SCP02/ICV The TSF shall perform **encryption and decryption of message integrity ICV** in accordance with a specified cryptographic algorithm **(Simple DES in ECB mode)** and cryptographic key sizes **(56 bits)** that meet the following: **FIPS 46-2**.

FCS_COP.1-DAP Cryptographic operation

FCS_COP.1.1-DAP The TSF shall perform **verification of the DAP signature attached to Executable Load Files** in accordance with a specified cryptographic algorithm **(RSA SSA-PKCS1-v1.5)** and cryptographic key sizes **of 1024 bits** that meet the following: **PKCS#1**.

FCS_COP.1-SCP02/FINAL Cryptographic operation

FCS_COP.1.1-SCP02/FINAL The TSF shall perform **MAC verification of the messages exchanged through a secure channel** in accordance with a specified cryptographic algorithm **(single DES plus final Triple DES)** and cryptographic key sizes **(16 bytes)** that meet the following: **FIPS PUB 46-3, ISO 9797-1**.

Application note:

Single DES plus final Triple DES is known is defined in [ISO 9797-1] as MAC Algorithm 1 without output transformation 3, without truncation, and with DES taking the place of the block cipher.

Appendixes B, D and E of [GPCS] specify the padding and ICV to be used for each of the abovementioned cryptographic operations in GlobalPlatform 2.1.1.

Key Loading Services

This section introduces the requirements for loading or replacing the static keys used to implement a Secure Channel. This mainly concerns the processing of the PUT KEY commands sent to the ISD.

FCS_CKM.3-KL Cryptographic key access

FCS_CKM.3.1-KL The TSF shall perform **ISD key loading and replacement** in accordance with a specified cryptographic key access method **(through a PUT KEY or STORE DATA command)** that meets the following: **[GPCS] and [VGP]**.

FPT_TDC.1-KL Inter-TSF basic TSF data consistency

FPT_TDC.1.1-KL The TSF shall provide the capability to consistently interpret **the Isuer Security Domain keys** when shared between the TSF and another trusted IT product.

FPT_TDC.1.2-KL The TSF shall use **the following rules:**

- **decryption of the data field of the received command shall be performed using the secure channel encryption session key**
- **before being set, imported DES key values shall be decrypted using the data encryption session key**

when interpreting the TSF data from another trusted IT product.

FDP_ITC.1-KL Import of user data without security attributes

FDP_ITC.1.1-KL The TSF shall enforce the **Secure Channel Protocol (SCP)** when importing user data, controlled under the SFP, from outside of the TSC.

FDP_ITC.1.2-KL The TSF shall ignore any security attributes associated with the user data when imported from outside the TSC.

FDP_ITC.1.3-KL The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TSC: **if the imported data is a key set for the ISD, then:**

- **If the command specifies the replacement of a key set, the specified key set must exists on the card, it must have the same number of components than the proposed new one, and each component of the existing key must have the same type and length than the proposed new one.**

- **The type of the imported keys shall be supported by the configuration of GlobalPlatform specified in [VGP] (DES keys)**
- **The data field of the command shall be decrypted using the data encryption key**
- **If the specified keys are DES keys, then the attached check value shall be correct**.

Key Generation

This section specifies the cryptographic requirements concerning the generation and distribution of the session keys used for setting up the Secure Channel Protocol.

FCS_CKM.1-SCP-SESSION-KEYS Cryptographic key generation

FCS_CKM.1.1-SCP-SESSION-KEYS The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **(the session key generation algorithm described in Appendix E.4.1 of [GPCS])** and specified cryptographic key sizes **(16-bytes key)** that meet the following: **ANSI X9.52 and ISO 10116**.

Application note:

The algorithm used to generate the session key includes the encryption of a derivation data depending on the session using a Triple DES in EBC mode. Such encryption algorithm shall follow the standards cited in this component.

FCS_CKM.2-SCP-SESSION-KEYS Cryptographic key distribution

FCS_CKM.2.1-SCP-SESSION-KEYS The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method **(the SCP02 cryptographic protocol)** that meets the following: **[GPCS]**.

Application note:

During the set up of a Secure Channel, the TOE agrees with the card host on the session key to be used for that channel. The TOE never distributes the session key (in the sense of sending the whole key to the host) but it exchanges information with the host that enable each party to derive the session key on its own side. This requirement comes form [JCSPP]. It has been placed in the Card Management section for the sake of clearness.

FMT_MSA.2-KEYS Secure security attributes

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

Application note:

Before using a key, the platform shall check that the operation to be performed with it is in accordance with its secure attributes, like its length, associated algorithm (DES, RSA, etc) and key type (public, secret).

Key Destruction Services

This section introduces the requirements to be met when deleting the cryptographic keys used for card management.

FCS_CKM.4-KD Cryptographic key destruction

FCS_CKM.4.1-KD The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method **[information removed]** that meets the following: **[information removed]**.

6.1.1.3. Card Content Management

This section introduces the security policy concerning the loading, installation and deletion of applications on the card. That function controls the operations of the ISD security policy regarding the dispatching of the INSTALL, LOAD and DELETE commands introduced in [VGP]. It also specifies requirements concerning the atomicity of those operations modifying the card contents, and the correct reception and interpretation of the code and the privileges of a new application.

Another goal of this security policy is to confirm that the Card Administrator has performed the off-card verifications ensuring that the imported piece of code is harmless, and that it has not been modified afterwards.

The requirements in this section correspond to the *Installer* group of requirements in [JCSPP].

FDP_UIT.1-CCM Data exchange integrity

FDP_UIT.1.1-CCM The TSF shall enforce the **Secure Channel Protocol (SCP) information flow control policy and the Issuer Security Domain (ISD) access control policy** to be able to **receive** user data in a manner protected from **deletion, replay, modification and insertion** errors.

FDP_UIT.1.2-CCM The TSF shall be able to determine on receipt of user data, whether **deletion, modification, replay and insertion** has occurred.

Application note:

The user data to be protected are the new Executable Files received by the card.

This Security Target refines the FDP_UIT.1/CM requirement introduced in [JCSPP] by requesting the enforcement of the ISD access control policy. Executable Files are sent as a sequence of LOAD commands. The SCP policy protects each single LOAD command from being modified during its transmission to the card. The ISD policy prevents the LOAD commands sent to the card from being inserted, replayed or deleted, see Rule 6 of that policy.

FDP_ROL.1-CCM Basic rollback

FDP_ROL.1.1-CCM The TSF shall enforce **Issuer Security Domain access control policy (ISD)** to permit the rollback of the **installation operation** on the **Executable Files and application instances**.

FDP_ROL.1.2-CCM The TSF shall permit operations to be rolled back within the **following boundary limit: until the Executable File or application instance has been added to the applet's registry**.

Application note:

The platform shall be able to safely abort the loading of a new Executable File whenever a corrupted or out-of-sequence LOAD command is received, irrespectively to the length of the file being loaded.

FDP_ITC.2-CCM Import of user data with security attributes

FDP_ITC.2.1-CCM The TSF shall enforce the **Firewall access control policy and the Secure Channel Protocol information flow policy** when importing user data, controlled under the SFP, from outside of the TSC.

FDP_ITC.2.2-CCM The TSF shall use the security attributes associated with the imported user data.

FDP_ITC.2.3-CCM 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-CCM 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-CCM The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TSC:

- **An Executable Load File may depend on (import or use data from) other Executable Files already installed on the card. This dependency is explicitly stated in the Executable Load File in the form of a list of Executable File AIDs. The loading is allowed only if, for each dependent Executable File, its AID attribute is equal to a resident Executable File AID attribute, and the major (minor) Version attribute associated to the former is equal (less than or equal) to the major (minor) Version attribute associated to the latter ([JCVM], §4.5.2). The intent of this rule is to ensure the binary compatibility of the package with those already on the card ([JCVM], §4.4)**.

FPT_FLS.1-CCM Failure with preservation of secure state

FPT_FLS.1.1-CCM The TSF shall preserve a secure state when the following types of failures occur: **the Issuer Security Domain fails to load/install an Executable File / application instance as described in [JCRE]**.

FPT_RCV.3-CCM/ELF Automated recovery without undue loss

FPT_RCV.3.1-CCM/ELF When automated recovery from **a failure or service discontinuity** 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-CCM/ELF For **abortion of the installation process of an Executable Load File, detection of a potential loss of integrity during the transmission of an Executable Load File to the card, and any fatal error occurred during the linking of an Executable Load File to the Executable Files already installed on the card**, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

FPT_RCV.3.3-CCM/ELF The functions provided by the TSF to recover from failure or service discontinuity shall ensure that the secure initial state is restored without exceeding **the loss of the Executable Load File being installed** for loss of TSF data or objects within the TSC.

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

FPT_RCV.3-CCM/AI Automated recovery without undue loss

FPT_RCV.3.1-CCM/AI When automated recovery from **a failure or service discontinuity** 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-CCM/AI For **abortion or any error or exception thrown during the installation process of a new application instance**, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

FPT_RCV.3.3-CCM/AI The functions provided by the TSF to recover from failure or service discontinuity shall ensure that the secure initial state is restored without exceeding **the loss of the Java Card objects created during the installation of the new Application instance** for loss of TSF data or objects within the TSC.

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

FRU_RSA.1-CCM Maximum quotas

FRU_RSA.1.1-CCM The TSF shall enforce maximum quotas of the following resources: **imported Executable Files and declared classes, methods and fields** that **subjects** can use **simultaneously**.

Application note:

The term "subjects" refer here to the Executable Files (or Packages, in Java Card jargon) that import or declare the listed resources.

6.1.1.4. Global Cardholder Verification Method

This section introduces the security requirements concerning GlobalPlatform's Cardholder Verification Method (CVM). This policy controls the modification of the internal data structures of the CVM that can be performed by the installed applications through the GlobalPlatform API.

FIA_AFL.1-CVM Authentication failure handling

FIA_AFL.1.1-CVM The TSF shall detect when **an administrator configurable positive integer within 1 and 255** unsuccessful authentication attempts occur related to **the authentication of the Cardholder**.

FIA_AFL.1.2-CVM When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall **temporarily lock the Cardholder authentication service, until an unlocking action has been successfully undertaken by a privileged user**.

Application note:

The PIN services created by the applications instances remain locked until the applet that owns the PIN object resets its state through the Java Card API.

6.1.2. Runtime Environment

This section contains the security functional requirements concerning the Runtime Environment that executes the applets. The requirements are gathered into the following categories:

- Core Requirements, which corresponds to the *CoreG* group of requirements specified in [JCSPP].
- Applet Deletion, which corresponds to the *ADEL* group of requirements specified in [JCSPP].
- Garbage Collection, which correspond to the *ODEL* group of requirements specified in [JCSPP].
- Defensive Virtual Machine requirements, which specify an additional access control policy that jTOP's virtual machine enforces at runtime.
- Operating System Requirements, which corresponds to a subset of the *Secure Card Platform* group of requirements in [JCSPP], namely, those concerning the implementation of the Operating System of the card.

6.1.2.1. Core Requirements

This group is focused on the main security policy of the Java Card System, known as the Firewall. This policy essentially concerns the security of installed applets.

Firewall Policy

FDP_ACC.2-FIREWALL Complete access control

FDP_ACC.2.1-FIREWALL The TSF shall enforce the **FIREWALL access control policy** on **the application instances and the JCRE (subjects), the Java Card class instances and arrays (objects)** and all operations among subjects and objects covered by the SFP.

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

Application note:

The operations under the control of the Firewall policy include:

- Reading and writing an array position or an instance field
- Invoking a virtual method on a class instance or an array
- Invoking an interface method on a class instance
- Throwing a Java Card exception
- Comparing the class of a class instance or an array against a given class
- Allocating a new class instance or array

See [JCSPP] for a detailed description of the subject, objects and operations under the control of the Firewall policy.

FDP_ACF.1-FIREWALL Security attribute based access control

FDP_ACF.1.1-FIREWALL The TSF shall enforce the **FIREWALL access control policy** to objects based on the following: **the security attributes for subjects and objects defined in [JCSPP] for this security policy**.

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: **the access control rules stated in [JCSPP] for this security policy**.

FDP_ACF.1.3-FIREWALL The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **the explicit access rules stated in [JCSPP] for this security policy**.

FDP_ACF.1.4-FIREWALL The TSF shall explicitly deny access of subjects to objects based on the **rules for explicit deny of access stated in [JCSPP] for this security policy**.

Application note:

The [JCSPP] introduces a detailed notation for defining the attributes and access rules for the Firewall policy. The detailed version of the rules is not repeated here for the sake of conciseness, but the following paragraphs provides a short summary of it.

An application instance has the following security attributes:

- The *Active Context* to which the applet instance belongs. Two instances of an applet declared in the same Java Card package belong to the same context.
- The *Selected Applet Context*, stating whether the instance is currently selected for execution.
- The *Currently Active Context*, which states what is the instance that is currently executing a bytecode. This attribute could be rather considered as TSF data supporting the rules of the policy.
- The *Multiselectable* attribute indicates whether the applet instance may be selected on several logical channels at the same time
- The *ActiveApplets* attribute lists all the applet instances declared in the same Java Card package as this one that are currently selected on a logical channel.

A class instance or array has the following attributes:

- Its sharing type, which classifies them into the following categories:
  - Standard objects, 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 point objects, who have freely accessible methods but protected fields,
  - Global arrays, having both freely access positions and methods.
- Its owner, which is the applet instance that allocated the class instance or array.
- The lifetime of the information it contains, which may be either cleared when the last active applet in the contexts of its owner is deselected, or when the card is reset.

The Firewall policy introduce the following rules to determine the access to an object:

- Rule 1: The following operations may be performed on an object that is a JCRE Entry Point:
  - Reading or writing its instance fields or array positions
  - Invoking a virtual or interface method
  - Throwing the object as an exception
  - Comparing the class of the object against a given one
- Rule 2: An applet instance may perform any of the operations under the control of the policy on a JCRE Entry Point object or a global array.
- Rule 3: An applet instance may perform any of the operations under the control of the policy on the persistent standard objects owned by an applet instance belonging to the same context.
- Rule 4: An applet instance may perform the following operations on a persistent object owned by an applet instance belonging to a different context:
  - Invoking an interface method
  - Comparing the class of the object against a given class
- Rule 5: Any applet instance may create standard persistent objects

In addition to the conditional rules above, the Firewall policy also introduces a special rule for explicit access:

- Rule 6: The JCRE may perform any of the operations on any persistent object, and on any object containing transient information to be cleared when the card is reset.

Finally, the Firewall policy enforces the following rule for explicit denial of access:

- Rule 7: No subject different from the currently selected applet is allowed to perform an operation on objects containing transient information to be cleared on deselection of the current applet. This rule concerns both the operation that creates an object and the operation that access an existing object.
- Rule 8: An applet instance cannot invoke an interface method on a class instance that is owned by an applet instance belonging to a different context when this latter applet instance is not multi-selectable and is currently active on another logical channel. This rule is introduced by the *Logical Channels* group of requirements of [JCSPP].

FDP_IFC.1-JCVM Subset information flow control

FDP_IFC.1.1-JCVM The TSF shall enforce the **JCVM information flow control policy** on **the subjects, information and operations defined in [JCSPP] for this security policy**.

Application note:

The information under the control of the JCVM information flow policy is the JCRE Temporary Entry point objects. The policy prevents those objects from being transferred from temporary storage (the operand stack, the local variables of a method) to persistent ones (static or instance fields, array positions).

See [JCSPP] for a detailed description of the subject, objects and operations under the control of this policy.

FDP_IFF.1-JCVM Simple security attributes

FDP_IFF.1.1-JCVM The TSF shall enforce the **JCVM information flow control policy** based on the following types of subject and information security attributes: **those defined in [JCSPP] for this security policy**.

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: **those defined in [JCSPP] for this security policy**.

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

FDP_IFF.1.4-JCVM The TSF shall provide the following **list of additional capabilities: none**.

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

FDP_IFF.1.6-JCVM The TSF shall explicitly deny an information flow based on the following rules: **when one of the conditions in the element FDP_IFF.1.1-JCVM above is not satisfied**.

Application note:

This Security Target does not mandate any additional access control rules with respect to the JCVM security policy defined in [JCSPP]. The components for specifying additional rules for explicitly granting or denying access are therefore assigned with "none". The component for specifying additional capabilities is also assigned with "none".

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**.

FMT_MSA.1-JCRE Management of security attributes

FMT_MSA.1.1-JCRE The TSF shall enforce the **FIREWALL access control policy and the JCVM information flow control policy** to restrict the ability to **modify** the security attributes ***Active Context***, ***ActiveApplets***, **and *Selected Applet Context*** to **the JCRE**.

Application note:

In Java Card specifications, the JCRE appears as the subject in charge of modifying the currently selected applets, see the chapter *Logical Channels and Applet Selection* of [JCRE].

In a smart card compliant with GlobalPlatform's specifications, the OPEN should therefore be considered as being part of the JCRE, as it is the subject in charge or selecting which is the applet instance that will be selected for the current session.

FMT_SMF.1-FIREWALL Specification of management functions

FMT_SMF.1.1-FIREWALL The TSF shall be capable of performing the following security management functions: **initializing the *Active Context* and the *Selected Applet Context* and performing a context switch on the former when an instance method is invoked**.

FMT_MSA.2-JCRE Secure security attributes

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

FMT_MSA.3-FIREWALL Static attribute initialisation

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

FMT_MSA.3.2-FIREWALL 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_SMR.1-JCRE Security roles

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

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

FPT_SEP.1-FIREWALL TSF domain separation

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

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

Application note:

The term "security domain" is not used here with the technical meaning that GlobalPlatform attaches to it, but in the sense of an execution context separate from the one used for the applications instances, like the *JCRE execution context* defined in [JCRE].

Application Programming Interface

The following requirements are related to the services that the Runtime Environment offers to the applet instances through the Java Card API.

Application Cryptography Services

The following requirements describe the cryptoghraphy services provided in the Java Card API.

FCS_COP.1-APP-CIPHER/DES Cryptographic operation

FCS_COP.1.1-APP-CIPHER/DES The TSF shall perform **encryption and decryption of application instance's data** in accordance with a specified cryptographic algorithm **(Triple DES either in CBC or ECB mode and with or without padding)** and cryptographic key sizes **of 112 bits** that meet the following: **FIPS PUB 46-3, FIPS PUB 81, ISO 9797, Java Card Application Programming Interface**.

FCS_COP.1-APP-SIGN/DES Cryptographic operation

FCS_COP.1.1-APP-SIGN/DES The TSF shall perform **signature generation and verification of application instance's data** in accordance with a specified cryptographic algorithm **(4-bytes or 8-bytes long MAC using Triple DES in CBC mode and with or without padding)** and cryptographic key sizes **of 112 bits** that meet the following: **FIPS PUB 46-3, ISO 9797, Java Card Application Programming Interface**.

FCS_COP.1-APP-SHA Cryptographic operation

FCS_COP.1.1-APP-SHA The TSF shall perform **computation of a hash value for application instance's data** in accordance with a specified cryptographic algorithm **(SHA-224 and SHA-256)** and cryptographic key sizes **none** that meet the following: **FIPS 180-2**.

FCS_COP.1-APP-RIPMD Cryptographic operation

FCS_COP.1.1-APP-RIPMD The TSF shall perform **computation of a hash value for application instance's data** in accordance with a specified cryptographic algorithm **(RIPMD 160)** and cryptographic key sizes **none** that meet the following: **ISO 10118-3**.

FCS_COP.1-APP-RSA Cryptographic operation

FCS_COP.1.1-APP-RSA The TSF shall perform **signature generation, signature verification, encryption and decryption on application instance's data** in accordance with a specified cryptographic algorithm **(RSA)** and cryptographic key sizes **(all multiples of 32 bits from 1536 bits, and up to 2048 bits)** that meet the following: **PKCS#1.5**.

FCS_RND.1-APP Quality metric for random numbers

FCS_RND.1.1-APP The TSF shall provide a mechanism to generate random numbers that meet **the *STANDARD* level specified in [DCSSI2791]**.

Key Generation

The TOE shall support on-card generation of different types of cryptographic keys.

FCS_CKM.1-APP-RSA Cryptographic key generation

FCS_CKM.1.1-APP-RSA The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **[information removed]** and specified cryptographic key sizes **(1536 to 2048 bits)** that meet the following: **[information removed]**.

Application note:

Annex A of the IEEE P1363-2000 standard (*Number Theoretic Background*) specifies the Miller-Rabin test that shall be used for determining whether the key parts that the random number generator yielded are prime numbers, with an arbitrary small probability of error. For the specified lengths, the keys generated shall be in the RSA-CRT format.

FCS_CKM.1-APP-EC Cryptographic key generation

FCS_CKM.1.1-APP-EC The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **(Elliptic Curves private keys respecting a given EC domain and curve)** and specified cryptographic key sizes

(160, 192, 224 or 256 bits) that meet the following: **ISO/IEC 15946-1, ISO/IEC 15946-3 and BSI's TR-03110**.

FCS_CKM.1-APP-DH Cryptographic key generation

FCS_CKM.1.1-APP-DH The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **(generating Diffie-Hellman keys using a random number generator and perfoming a modular exponential according to a given domain)** and specified cryptographic key sizes **(1536 to 2048 bits)** that meet the following: **PKCS#3**.

Key Agreement

The TOE supports several key agreement mechanisms based on the Diffie-Hellman protocol. This protocol enables the smart card to agree with card host on a shared secret that can be used to derive session keys. The TOE is not expected to directly distribute the session key (in the sense of sending the key value to the host) but it exchanges information with the host that enable each party to derive a shared secret on its own side.

The following SFR specify several methods for generating a shared secret that provides a suitable input for deriving a triple DES session key to be used in the Diffie-Hellman protocol, for instance using the method described in §4.3.3 (3DESKDF) of ISO/IEC 15946-3. However, this method is not intended to be the only possible method that an applet may use to derive a triple DES key. Actually, the Java Card Technology does not provide any specific service for creating cryptography key values of DES type, it just provides the means to create secure containers to hold such values. The DES keys that the applet places in such containers are ultimately provided by the Card Administrator or generated by the applet itself. The method used to create or import them is the responsibility of the TOE environment, which is subject to the OSP.SECRETS organizational policy.

FCS_CKM.2-APP-DH-EC Cryptographic key distribution

FCS_CKM.2.1-APP-DH-EC The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method **(Diffie-Hellman key agreement based on an elliptic curve cryptography algorithm)** that meets the following: **IEEE P1363**.

FCS_CKM.2-APP-DH-EG Cryptographic key distribution

FCS_CKM.2.1-APP-DH-EG The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method **(Diffie-Hellman key agreement based on El Gamal algorithm)** that meets the following: **ISO 15946-1, ISO 15946-3 and BSI's TR03110.**.

Application note:

The relevant section of ISO/IEC 15946-3 is §3.2.1 (Key agreement of ElGamal-type).

Residual Information Protection

The following requirements concern the protection of sensitive residual information that could be left in Java Card objects.

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**.

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:

The `bArray` object is the byte array passed as real argument of the `Applet.install()` method when a new applet instance is created.

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**.

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)**.

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 Java Card transient object**.

Application note:

The events that provoke the de-allocation of any transient object are described in [JCRE].

FDP_ROL.1-FIREWALL Basic rollback

FDP_ROL.1.1-FIREWALL The TSF shall enforce **the FIREWALL access control policy and the JCVM information flow control policy** to permit the rollback of the **creation and access operations through Java Card bytecodes** on the **Java Card objects**.

FDP_ROL.1.2-FIREWALL The TSF shall permit operations to be rolled back within the **scope of a `select()`, `deselect()`, `process()` or `install()` call, notwithstanding the restrictions given in [JCRE], within the bounds of the commit capacity and those described in [JCAPI]**.

Card Security Management

The following requirements are related to the security of the whole card, in contrast to the previous ones, that are somewhat restricted to the features of the Runtime Environment alone. For instance, a potential security violation detected by the virtual machine may require a reaction that does not only concern the virtual machine, such as requesting the appropriate security module with the power to block the card to perform the operation.

FAU_ARP.1-JCS Security alarms

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

- **throwing an exception;**
- **locking the card session (card muting);**
- **reinitializing the Java Card System and its data;**
- **temporary disabling the services of the card until a privileged roles performs a special action;**
- **definitely disabling all the services of the card**

upon detection of a potential security violation.

Global refinement:

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

- CAP file inconsistency
- Typing error in the operands of a bytecode
- Applet life cycle inconsistency
- Card tearing (unexpected removal of the Card out of the CAD) and power failure
- Abortion of a transaction in an unexpected context
- Violation of the Firewall or JCVM security policies
- Unavailability of resources
- Array overflow
- Other runtime errors related to applet's failure, like uncaught exceptions.

FDP_SDI.2 Stored data integrity monitoring and action

FDP_SDI.2.1 The TSF shall monitor user data stored within the TSC for **integrity errors on cryptography keys and PIN values** on all objects, based on the following attributes: **a checksum on the value of those objects**.

FDP_SDI.2.2 Upon detection of a data integrity error, the TSF shall **mute the card if an application attempts to use the corrupted key or PIN.**.

FPT_RVM.1 Non-bypassability of the TSP

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

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, their bytecodes and their data arguments** when shared between the TSF and another trusted IT product.

FPT_TDC.1.2 The TSF shall use **the following rules:**

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

when interpreting the TSF data from another trusted IT product.

FPT_FLS.1-JCS Failure with preservation of secure state

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

FPR_UNO.1-CRYPTO Unobservability

FPR_UNO.1.1-CRYPTO The TSF shall ensure that **external users or malicious application instances fraudulently installed on the card** are unable to observe the operation **encryption, decryption, signature generation and verification** on **application instance's data** by **application instances**.

Application note:

The ISD shall be considered as a distinguished application instance which is therefore covered by this requirement.

FPR_UNO.1-PIN Unobservability

FPR_UNO.1.1-PIN The TSF shall ensure that **external users or malicious application instances fraudulently installed on the card** are unable to observe the operation **comparison** on **PIN codes** by **application instances**.

FPT_TST.1 TSF testing

FPT_TST.1.1 The TSF shall run a suite of self tests **during initial start-up** to demonstrate the correct operation of the TSF. operation of **the TSF depending on specific support from the IC**.

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

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

Global refinement:

The initial start-up corresponds to power on.

AID Management

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 and the Issuer Security Domain**.

Application note:

The only modification that is allowed are extending the set of registered application instances with a new one, or deleting an existing application instance.

FMT_MTD.3-AID Secure TSF data

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

Application note:

This requirement concerns the use of valid Application IDentifiers (AID). An AID is valid if it is unique, has the right length, and was specified in the INSTALL command that created the Executable File or applet instance that it identifies.

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: **the AID and version number of each Executable File, the AID of each registered applet, and whether a registered applet is currently selected for execution or active on some logical channel.**.

FIA_UID.2-AID User identification before any action

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

Application note:

This requirement refers to the identification of the application instances when they request a service from the TOE that is under the control of one of the TSP defined in this Security Target. In particular, the access control rules of the Firewall and the Cardholder Verification Method requires identifying the AID of the application instance that requires access to a Java Card object or to the global PIN of the card.

FIA_USB.1 User-subject binding

FIA_USB.1.1 The TSF shall associate the following user security attributes with subjects acting on the behalf of that user: **the active context of each application instance**.

FIA_USB.1.2 The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: **when an instance of an applet class declared in a Java Card package P is created, that package P is taken as the active context associated to the new application instance**.

FIA_USB.1.3 The TSF shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users: **none**.

Application note:

As this Security Target is compliant with version 2.2 of the Common Criteria, this requirement has been refined with the additional elements that this version adds to this component.

The access control rules of the Java Card Firewall make use of *currently active context* to decide whether access to a Java Card object can be granted to an application instance. All the instances of an Applet class that is declared inside the same Java Card package belong to the same active context and have the same access rights to Java Card objects. See [JCRE] for a detailed explanation of the notion of active context.

Applet Deletion Manager Policy

FDP_ACC.2-ADEL Complete access control

FDP_ACC.2.1-ADEL The TSF shall enforce the **ADEL access control policy** on **the Applet Deletion Manager (subject), the loaded Executable Files, the installed applet instances and their class instances and arrays (objects),** and all operations among subjects and objects covered by the SFP.

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

Application note:

The operations under the control of the ADEL policy include:

- Deleting an applet instance installed on the card;
- Deleting an Executable File loaded on the card.

In a smart card compliant with [VGP], the role of the *Applet Deletion Manager* is played by the Issuer Security Domain, and the deletion operations are performed through the DELETE APDU command. See [JCSPP] for a detailed description of the subject, objects and operations under the control of this policy.

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: **the security attributes defined in [JCSPP] for this security policy**.

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: **the access control rules stated in [JCSPP] for this security policy**.

Non editorial refinement:

The TOE allows the deletion of an applet instance event if they are objects owned by the applet instance that are referenced from the package that declares the applet. In this case, the TOE shall replace the references in the static fields by null values.

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 **the deny access rules defined in [JCSPP] for this security policy**.

Application note:

The [JCSPP] introduces a detailed notation for defining the attributes and access rules for the ADEL policy. The detailed version of the rules is not repeated here for the sake of conciseness, but the following paragraphs provides a short summary of it.

An application instance has the following security attributes:

- The *Selection State* specifies whether the applet is currently active on some logical channel.

A class instance or array has the following security attributes:

- The *Owner* specifies the applet instance that created the class instance or array;
- The *Class* specifies of which class the object is an instance of. This attribute is not explicitly mentioned in [JCSPP] but is implicit in the access control rules of the ADEL policy.

An Executable File has the following security attributes:

- The *Name* identifies the file;
- The *Imported Files* specifies the files on which it depends;
- The *Static References* specifies the class instances or arrays that are directly reachable from the file.

The ADEL policy introduce the following rules to delete an applet instance or Executable File:

- Rule ADEL-1: An applet instance may only be deleted by the Applet Deletion Manager, provided that the applet instance is not currently active, and that all its class instances and arrays are neither reachable from an Executable File (different from the one declaring the applet, according to the refinement made by ths ST), nor from other applet instances.
- Rule ADEL-2: An Executable File containing a library may only be deleted by the Applet Deletion Manager, provided that no other file depends on this one, and that there is no instance of a class defined in this file that is reachable from other files.
- Rule ADEL-3: An Executable File declaring Applet classes may only be deleted by the Applet Deletion Manager, provided that it fulfills the premises in Rule 2, and that all the instances of the Applets declared in it satisfy the premises in Rule 1.

The ADEL policy in [JCSPP] contains an extra rule concerning the deletion of a collection of applet instances in a single step (rule R.JAVA.15, *"Multiple Applet Instance Deletion"*). That rule does not apply to a TOE compliant with GlobalPlatform, as this standard does not support multiple applet deletion.

Finally, the ADEL policy explicitly denies deletion access to other subjects different from the Applet Deletion Manager.

FMT_MSA.3-ADEL Static attribute initialisation

FMT_MSA.3.1-ADEL The TSF shall enforce the **ADEL access control policy** 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_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.

Application note:

In a smart card compliant with [VGP], the role of the *Applet Deletion Manager* is played by the Issuer Security Domain.

6.1.2.2. Defensive Virtual Machine

This group of requirements concerns the dynamic verifications of the applet's bytecode that jTOP's Java Card Virtual Machine implements. Some of these verifications are redundant with respect to the BCVG group of security requirements that [JCSPP] imposes on the IT environment. They constitute a second line of verifications for counter those attacks based on the execution of ill-typed applications.

FDP_ACC.1-DVM Subset access control

FDP_ACC.1.1-DVM The TSF shall enforce the **Defensive Virtual Machine (DVM) access control policy** on **the following list of subjects, objects and operations:**

- **The subjects under the control of the DVM security policy are the Application instances installed on the card.**
- **The objects under the control of the DVM security policy are:**
  - **the Java Card class instances and arrays**
  - **the code of the Executable Files**
  - **the static data of the Executable Files (static field image)**
  - **the runtime data areas of the JCVM: the operand stack and local variables of the currently executed method**
- **The operations under the control of the DVM security policy are the Java Card bytecodes**.

FDP_ACF.1-DVM Security attribute based access control

FDP_ACF.1.1-DVM The TSF shall enforce the **Defensive Virtual Machine (DVM) access control policy** to objects based on the following: **[Information removed]**.

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

- **Rule DVM-1: An applet may pop values from the operand stack only if there are enough values on the stack to perform such operation.**
- **Rule DVM-2: An applet may push values onto the operand stack only if there is enough room on the operand stack for the new values.**
- **Rule DVM-3: An applet may read or write a local variable of a given method only if the local variables is within the ones that the applet declared for this method.**

[Information removed].

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

FDP_ACF.1.4-DVM The TSF shall explicitly deny access of subjects to objects based on the **following rule: when none of the rules listed in the element FDP_ACF.1.1 of this component is satisfied**.

FMT_MSA.3-DVM Static attribute initialisation

FMT_MSA.3.1-DVM The TSF shall enforce the **Defensive Virtual Machine (DVM) access control policy** to provide **restrictive** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2-DVM The TSF shall allow the **following roles: none of them** to specify alternative initial values to override the default values when an object or information is created.

6.1.2.3. Applet Deletion

This group of requirements concerns the mechanism for deleting Executable Files and applet instances introduced in the versions 2.2.x of Java Card. They correspond to the *ADEL* group of requirements in [JCSPP].

Additional Deletion Requirements

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 Executable Files when one of the deletion operations in FDP_ACC.2.1-ADEL is performed on them**.

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 an Executable File or applet instance as described in [JCRE]**.

6.1.2.4. Garbage Collection

This group of requirements concerns the mechanism for on-demand garbage collecting unreachable objects that was introduced in the versions 2.2.x of Java Card. They correspond to the ODEL group of requirements in [JCSPP].

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 that triggered the execution of the `JCSystem.requestObjectDeletion()` method**.

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 `JCSystem.requestObjectDeletion()` method**.

Application note:

Unreachable objects are deleted during the card booting phase, provided that a GC round has been previously booked by some applet. The boot phase has strong constraints in terms of execution time, as the card has to provide the terminal with an answer. Should the card spend all the time available for removing unreachable objects, it shall ensure that broken references are not introduced into reachable ones.

6.1.3. Smart Card Platform

The *Smart Card Platform* group introduced in [JCSPP] specifies the IT requirements that are imposed on the Operating System and the Integrated Circuit underlying the implementation of the Runtime Environment. Because of the modification in the scope of evaluation, which does include in this Security Target the Operating System and the Integrated Circuit, those requirements on the IT environment become requirements on the TOE itself.

6.1.3.1. Operating System

This section presents those requirements of the *Smart Card Platform* group that concern the Operating System.

FPT_AMT.1-OS Abstract machine testing

FPT_AMT.1.1-OS The TSF shall run a suite of tests **during initial start-up** to demonstrate the correct operation of the security assumptions provided by the abstract machine that underlies the TSF.

Application note:

Start up means here the very first time that the smart card is powered on, and possibly subsequent power-ups. The underlying abstract machine refers to the Operating System and the Integrated Circuit upon which the Runtime Environment is constructed.

FPT_RCV.3-OS Automated recovery without undue loss

FPT_RCV.3.1-OS When automated recovery from **security policy violation** 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-OS For **execution access to a memory zone reserved for TSF data, writing access to a memory zone reserved for TSF's code, and any segmentation fault performed by a Java Card applet**, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

FPT_RCV.3.3-OS The functions provided by the TSF to recover from failure or service discontinuity shall ensure that the secure initial state is restored without exceeding

- **the contents of Java Card static fields, instance fields, and array positions that fall under the scope of an open transaction;**
- **the Java Card objects that were allocated into the scope of an open transaction;**
- **the contents of Java Card transient objects;**
- **any possible Executable Load File being loaded when the failure occurred**

for loss of TSF data or objects within the TSC.

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

FPT_RCV.4-OS Function recovery

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

6.1.3.2. Integrated Circuit

The section contains those requirements of the *Smart Card Platform* group introduced in [JCSPP] that concern the Integrated Circuit and those in [ICST]. Most of those requirements are left unassigned in the protection profile. They have been fulfilled in this Security Target with the ones specified for the chip in [ICST]. They mainly concern protecting the smart card's chip against physical tampering, preventing the disclosure of information when it is transferred from different physical parts of the chip, providing the basic DES operation, and keeping a secure state when a malfunction is detected and providing an independent security domain for the hardware.

Because of the enlargement in the scope of evaluation, the following requirements introduced in [ICST] for the non-IT environment are discharged in this Security Target by other functional requirements or assurance measures of the TOE:

- RE.Phase-1 on the correct design of the smart card's Embedded Software according to requirements of IC manufacturer. This requirement is covered by the assurance measures concerning the development of smart card embedded software and its environment.
- RE.Process-Card on the protection of the IC after its delivery. This requirement is covered by the assurance measures concerning the life cycle support of the Embedded Software.
- RE.Cipher on the usage of key-dependent functions. This requirement corresponds to all the requirements on the TOE concerning cryptographic support (FCS class), FDP_ITC.1-KL and FDP_ITC.1-SCP-SESSION-KEY, which ensure a correct management of keys and related functions.

FRU_FLT.2-IC Limited fault tolerance

FRU_FLT.2.1-IC The TSF shall ensure the operation of all the TOE's capabilities when the following failures occur: **exposure to operating conditions which are not detected according to the requirement FPT_FLS.1-IC**.

FPT_FLS.1-IC Failure with preservation of secure state

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

- **exposure to operating conditions when a malfunction could occur;**
- **failure detected by TSF according to FPT_TST.1-IC**.

FPT_SEP.1-IC TSF domain separation

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

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

Application note:

The parts of the TOE which support the security functional requirements FPT_FLS.1-IC and FPT_TST.1-IC should be protected from interference of the other security enforcing parts of the Embedded Software.

FPT_TST.2-IC Subet TOE testing

FPT_TST.2.1-IC The TSF shall run a suite of self tests **at the request of the authorized user** to demonstrate the correct operation of **the following environmental sensor mechanism: frequency monitoring, voltage sensor, light detection, temperature sensor, the RNG with help of the live test, the active shield**.

Application note:

The term "authorized user" used in [ICST] refers here to the Embedded Software.

FPT_PHP.3-IC Resistance to physical attack

FPT_PHP.3.1-IC The TSF shall resist **physical manipulation and physical probing** to the **TSF of the IC** by responding automatically such that the TSP is not violated.

Application note:

The IC will implement appropriate measures to continuously counter physical manipulation and physical probing. Due to the nature of these attacks (especially manipulation) the IC can by no means detect attacks on all of its elements. Therefore, permanent protection against these attacks is required ensuring that the TSP could not be violated at any time. Hence, "automatic response" means here (i) assuming that there might be an attack at any time and (ii) countermeasures are provided at any time.

FCS_COP.1-IC Cryptographic operation

FCS_COP.1.1-IC The TSF shall perform **encryption and decryption** in accordance with a specified cryptographic algorithm **Triple Data Encryption Standard (Triple DES)** and cryptographic key sizes **112 bits** that meet the following: **U.S. Department of Commerce - National Bureau of Standards Data Encryption Standard (DES), FIPS PUB 46-3, 1999 October 25, keying option 2**.

Application note:

The basic Triple DES primitive used to implement the other cryptographic operations described in the precedent sections must be provided by the smart card's chip.

FDP_ITT.1-IC Basic internal transfer protection

FDP_ITT.1.1-IC The TSF shall enforce the **Data Processing Policy** to prevent the **disclosure** of user data when it is transmitted between physically-separated parts of the TOE.

Global refinement:

The different memories, the CPU and other functional units of the TOE (e.g. a cryptographic co-processor) are seen as physically-separated parts of the TOE.

FPT_ITT.1-IC Basic internal TSF data transfer protection

FPT_ITT.1.1-IC The TSF shall protect TSF data from **disclosure** when it is transmitted between separate parts of the TOE.

Global refinement:

The different memories, the CPU and other functional units of the TOE (e.g. a cryptographic co-processor) are seen as separated parts of the TOE.

This requirement is equivalent to FDP_ITT.1-IC above but refers to TSF data instead of User Data. Therefore, it should be understood as to refer to the same Data Processing Policy defined under FDP_IFC.1-IC below.

FDP_IFC.1-IC Subset information flow control

FDP_IFC.1.1-IC The TSF shall enforce the **Data Processing Policy** on **on all confidential data when they are processed or transferred by the IC or the Embedded Software**.

Global refinement:

The following Security Function Policy (SFP) Data Processing Policy is defined in [SSVG] for the requirement FDP_IFC.1-IC:

"User Data and TSF data shall not be accessible from the IC except when the Embedded Software decides to communicate the User Data via an external interface. The protection shall be applied to confidential data only but without the distinction of attributes controlled by the Embedded Software."

Application note:

The term "processed" must be understood as "processed by the IC or the OS". Similarly, the term "transferred" must be understood as "internally transferred between the different parts of the IC".

FDP_ACC.1-MMU Subset access control

FDP_ACC.1.1-MMU The TSF shall enforce the **Memory Access Control Policy** on **all subjects (software running at system mode active during interrupt execution or application mode active during other executing), all objects (data including code stored in memories) and read, write, delete, and execute operations**.

Application note:

Usage of multiple applications in one smart card often requires separating code and data in order to prevent that one application can access code and/or data of another application. To support this the TOE provides Area based Memory Access Control.

FDP_ACF.1-MMU Security attribute based access control

FDP_ACF.1.1-MMU The TSF shall enforce the **Memory Access Control Policy** to objects based on the following:

- **Subjects:**
  - **software running at system mode active during interrupt execution or application mode active during other executing attributes:**
  - **the interrupt execution level where the software is executed (interrupt / non-interrupt) and/or**
- **Objects:**
  - **data including code stored in memories attributes:**
  - **the memory area where the access is performed to and/or**
  - **the operation to be performed.**.

FDP_ACF.1.2-MMU The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **evaluate the corresponding permission control information before the access so that accesses to be denied can not be utilized by the subject attempting to perform the operation**.

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

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

Application note:

Permission control shall prevent the subject from modifying a memory area storing the code of a TSF, or to execute a memory area containing TSF data.

FMT_MSA.3-MMU Static attribute initialisation

FMT_MSA.3.1-MMU The TSF shall enforce the **Memory Access Control Policy** to provide **well-defined (according to [ICDB], Section 5)** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2-MMU The TSF shall allow the **Java Card Runtime Environment** to specify alternative initial values to override the default values when an object or information is created.

Application note:

According to [ICST], the any subject is in principle allowed to modify the default values set for the MMU, and it is up to the Embedded Software to set the memory access control policy. In this Security Target, only the Java Card Runtime Environment is allowed to reprogram the MMU. On the contrary, applications are not allowed to modify the default values of the MMU at all.

FMT_MSA.1-MMU Management of security attributes

FMT_MSA.1.1-MMU The TSF shall enforce the **Memory Access Control Policy** to restrict the ability to **change_default, delete and modify** the security attributes **permission control information** to **running at interrupt level (system mode)**.

FMT_SMF.1-MMU Specification of management functions

FMT_SMF.1.1-MMU The TSF shall be capable of performing the following security management functions: **access the configuration registers of the MMU**.

6.2. TOE security assurance requirements

The security assurance requirement level is EAL5. The EAL is augmented with AVA_VLA.4 and ALC_DVS.2.

6.3. Security requirements for the IT environment

6.3.1. IT environment functional requirements

The only requirements that this Security Target specifies for the for the IT environment are the ones introduced in the *BCVG* group of [JCSPP].

6.3.1.1. BCVG Security Functional Requirements

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

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

FDP_IFC.2-BCV Complete information flow control

FDP_IFC.2.1-BCV The TSF shall enforce the **TYPING information flow control policy** on **the type of values flowing across local variables, operand stack positions, static and instance fields, array positions, and methods frames** and all operations that cause that information to flow to and from subjects covered by the SFP.

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

FDP_IFF.2-BCV Hierarchical security attributes

FDP_IFF.2.1-BCV The TSF shall enforce the **TYPING information flow control policy** based on the following types of subject and information security attributes: **the security attributes defined in [JCSPP] for this security policy**.

FDP_IFF.2.2-BCV The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules, based on the ordering relationships between security attributes hold: **the information flow rules defined in [JCSPP] for this security policy**.

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

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

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

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

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

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

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

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

FMT_MSA.1-BCV.1 Management of security attributes

FMT_MSA.1.1-BCV.1 The TSF shall enforce the **TYPING information flow control policy** to restrict the ability to **modify** the security attributes**: *Type* security attribute of the fields and methods** to **none**.

FMT_MSA.1-BCV.2 Management of security attributes

FMT_MSA.1.1-BCV.2 The TSF shall enforce the **TYPING information flow control policy** to restrict the ability to **modify** the security attributes**: *Type* security attribute of local variables and operand stack position** to **the role Bytecode Verifier**.

FMT_SMF.1-BCV Specification of management functions

FMT_SMF.1.1-BCV The TSF shall be capable of performing the following security management functions: **updating the type of the value stored in a local variable, operand stack position, static or instance field, and array position**.

FMT_MSA.2-BCV Secure security attributes

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

FMT_MSA.3-BCV Static attribute initialisation

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

FMT_MSA.3.2-BCV 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_SMR.1-BCV Security roles

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

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

FRU_RSA.1-BCV Maximum quotas

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

6.4. Security requirements for the non-IT environment

6.4.1. Non-IT environment functional requirements

This Security Target does not specify any security functional requirements for the non-IT environment.

7 TOE summary specification

7.1. TOE security functions

This chapter introduces the TOE Security Functions (TSF) that instantiate the security requirements introduced in the previous section. Each function is introduced providing its name and a summary of it in an informal style. A detailed description of each of them can be found in the document [FSP] and in the public specifications of GlobalPlatform [GPCS] and Java Card [JCVM], [JCRE], [JCAPI].

The minimum strength for the security functions is SOF-high.

7.1.1. Card Management

This section introduces the security functions that the GlobalPlatform layer provides.

7.1.1.1. Issuer Security Domain

The following security functions concern the security features of the Issuer Security Domain.

OPEN

This TSF is in charge of selecting the applet instances to be activated (selected for execution), and dispatching the received commands to them. It also initializes and manages the internal data structures required for implementing the card management services. In GlobalPlatform, this function is referred to as the OPEN (Open Platform ENvironment).

Card Content Management

This TSF controls the loading, installation, and removal of Executable Files and application instances.

Life Cycle Management

This TSF enforces the life cycle that GlobalPlatform defines for both the card and the installed applications in [GPCS]. It also controls that the card cannot shift from a GlobalPlatform state to one of the proprietary states of the Card Initialization phase.

Administration Commands Control

This TSF checks that only the card management commands specified in [FSP] are accepted, and rejects ill-formed ones with an appropriate error response. It also controls which card administration commands are allowed at each state of the smart card's life cycle.

7.1.1.2. Secure Channels

Most of card management duties involve importing or modifying security sensitive assets, like cryptographic keys or platform personalization data. The TOE uses the Secure Channel Protocol 02 (SCP02) specified in [GPCS] to protect such assets while they are in transit to the card, and to reject any piece of data coming from a distrusted user. This cryptographic protocol consists of three main steps: mutual authentication, message exchange and secure channel termination.

Host Authentication

This TSF enforces the authentication of the Card Administrator through the Secure Channel Protocol 02 (SCP02).

Message Integrity and Authentication

This TSF enforces the integrity and the origin of the APDU commands received through a Secure Channel.

Message Data Confidentiality

This TSF decrypts the contents of an APDU message containing sensitive information.

Secure Channel Termination

When a Secure Channel is closed this TSF ensures that the session keys are cleared and the security level is reset. Further commands received on the underlying logical channel are not considered as coming from the Card Administrator.

7.1.1.3. Secure Channel Key Management

The following security functions concern the keys required to set up a secure communication channels between the smart card and its host.

Session Key Generation

This TSF controls the generation of the session keys used to set up a Secure Channel with the CAD.

ISD Key Loading and Replacement

This TSF enables to load or replace one of the key sets that the ISD uses to establish a secure channel with the Card Administrator.

7.1.1.4. Cardholder Verification Management

The following security function concerns the management of a global service for authenticating the Cardholder.

Global CVM

This TSF controls the update of the security attributes associated to a global cardholder authentication service shared by all the applications installed on the platform.

7.1.2. Runtime Environment

This section introduces the security functions that the Runtime Environment provides.

7.1.2.1. Application Reference Monitors

The following security functions provide abstract machines that mediate the access to objects by subjects during the interpretation of the applet's bytecode.

Java Card Firewall

This TSF enforces the Firewall access control policy and the JCVM information flow control policy at runtime. The former policy controls object sharing between different applet instances, and between applet instances and the Java Card Runtime Environment. The latter policy controls the access to global data containers shared by all applet instances.

Defensive Java Card Virtual Machine

This TSF is a reference monitor on the actions that the application instances perform at runtime.

7.1.2.2. Security countermeasures

The following security functions concern the countermeasures that the TOE may trigger upon detection of a security policy violation.

Card Muting

This TSF makes the card to stop processing the current command and shifts it into a state where any further communication with it is impossible from outside until the next card reset (cold or warm).

When the card is muted, it does not react to any APDU command and sends no more answers to the CAD.

Card Locking

This TSF moves the card to a life cycle state where all applications except the Issuer Security Domain are disabled and cannot be selected for execution.

When the card is locked, its services are reduced to those offered to the Card Administrator, which require the authentication of this actor. In addition to this, the ISD rejects loading, installing or deleting any Executable File or Application instance until the card is unlocked.

Card Termination

This TSF moves the card to a state where all the servicesoffered by the smart card are definitely disabled, except for card management information retrieval. Any other information stored in the smart card such as keys or PIN codes is cleared.

When the card is terminated, applet selection is impossible and the card rejects any APDU command except GET DATA.

7.1.2.3. Life Cycle Management

The following security functions concern specific security actions that are performed upon special events in the life cycle of the platform.

Clearing sensitive information

This TSF clears all the data containers that hold sensitive information when that information is not longer used.

Booting Tests

This TSF checks at the beginning of each card session the correct initialization of all the security services, the integrity of the TSF data on which they relay. It also prevents the platform code from running on a chip different from the one for which it was designed.

7.1.2.4. Integrity

The following security functions contribute to enforce data integrity requirements.

Atomic Transactions

This TSF provides means to execute a sequence of modifications and allocations on the persistent memory so that either all of them are completed, or the card behaves as if none of them had been executed. The transaction mechanism is used for updating internal sensitive data as well as for performing different card management functions, such as installing a new Executable File on the card or creating a new applet instance.

7.1.2.5. Service Availability

The following functions contribute to prevent denial of service.

Resource Quotas

This TSF ensures that each application instance meets the memory quotas that the Card Administrator assigned to it when the instance was installed on the card. When the memory quota allocated for an application instance is exhausted, this TSF checks that no further memory is allocated to it. A maximum amount of memory may be also defined for all the instances of the classes declared in a given Executable File.

This TSF also checks that the Executable Files installed on the card comply with the CAP format constraints regarding the number of imported packages and the number of declared classes, instance fields and static fields that an application may define, according to [JCVM].

7.1.2.6. Cryptography

The following security functions implement the cryptography algorithms required for the platform.

Signature Generation and Verification

This TSF provides applets with a mechanism for generating an electronic signature of a byte array, and verifying an electronic signature stored in a byte array.

Signature algorithms are available to the applications through the `javacard.security.Signature` abstract class of the Java Card API, for which the Runtime Environment provides a concrete implementation.

Java Card Technology enables a given platform to implement a variable set of signature algorithms. The signature key types and algorithms from the Java Card API that the TOE supports are summarized in the reference [PROFILE].

If the applet specifies an algorithm that the platform does not support, the Runtime Environment refuses to create the requested signature instance. Some other signature algorithms, even though supported, do not fall into the scope of the TOE, and their use is not advised; see [USR].

Encryption and Decryption

This TSF provides the applets with a mechanism for encrypting and decrypting the contents of a byte array.

Ciphering algorithms are available to the applications through the `javacardx.Cipher` abstract class of the Java Card API, for which the Runtime Environment provides a concrete implementation.

Java Card Technology enables a given platform to implement a variable set of cipher algorithms. The key types and encryption algorithms from the Java Card API that the TOE supports are summarized in the reference [PROFILE].

Message Digest Generation

This TSF provides the application instances with a mechanism for generating an (almost) unique value for the contents of a byte array. Such a value can be used as a shorter available of the information contained in the whole byte array.

Message Digest algorithms are available to the applications through the `javacard.security.MessageDigest` abstract class of the Java Card API, for which the Runtime Environment provides a concrete implementation. The message digest algorithms from the Java Card API that the TOE supports are summarized in the reference [PROFILE].

Random Number Generation

This TSF provides the application instances with a mechanism for generating a randomly chosen piece of data. Such data is intended to be used for generating protocol challenges and key values.

Random number generators are available to the applets through the `RandomData` class of the Java Card API. The random numbers provided to the applets are compliant with DCSSI's "STANDARD" quality metric specified in [DCSSI2791].;

The strength of this function is SOF-high.

7.1.2.7. Key Management

The following security functions implement key infrastructure requirements.

Key Generation

This TSF provides the applets with on-card generation of cryptographic keys.

It only supports generation of RSA and EC key pairs. Key components are generated using a secure random number generator compliant with DCSSI's *Standard* security level for cryptography operations.

The strength of this function is SOF-high.

Key Agreement

This TSF enables an applet to agree on a shared secret with the terminal, without disclosing this secret to third parties that could observe the messages exchanged between the card and the terminal.

Java Card Technology enables a given platform to implement a variable set of key agreement algorithms. The key types and algorithms from the Java Card API that the TOE supports are summarized in the reference [PROFILE].

Key Encryption

This TSF protects the cryptographic keys from being read out from the memory. It ensures that keys are stored into secure containers in an encrypted format and provides the functions for accessing and modifying them in a secure container.

Key Integrity

This TSF checks the integrity of the keys before performing any cryptographic operation with them.

Key Destruction

This TSF disables the use of a key both logically and physically.

7.1.2.8. Cardholder Authentication

The following functions implement the requirements concerning the authentication of the Cardholder.

Cardholder Verification

This TSF enables applet instances to authenticate the sender of a request as the genuine Cardholder. Applets have access to this service through the `OwnerPIN` class of the Java Card API.

The strength of this function is SOF-high.

PIN Value Integrity

This TSF checks the integrity of the PIN and its persistent attributes (try-counter, try-limit and CVM state) before each Cardholder authentication attempt.

7.1.3. Integrated Circuit TSFs

This section recalls those security functions from [ICST] that are relevant for the security of this TOE.

Operating State Checking

This is the security function SEF1 "Operating state checking" introduced in [ICST].

Phase Management

This is the security function SEF2 "Phase Management with test mode lock-out" introduced in [ICST].

Protection Against Snooping

This is the security function SEF3 "Protection against snooping" introduced in [ICST].

Hardware Data Encryption

This is the security function SEF4 "Data Encryption and data disguising" introduced in [ICST].

Application note:

This security function supports the SF.Confidentiality one by providing low-level encryption of sensitive data.

True Random Number Generation

This is the security function SEF5 "Random Number Generation" introduced in [ICST].

The strength of this function is SOF-high.

Application note:

This security function supports the security function SF.RandomNumbers by providing the input seed for the software random number generator.

Hardware Self Test

This is the security function SEF6 "TSF Self Test" introduced in [ICST].

Notification of Physical Attack

This is the security function SEF7 "Notification of Physical Attack" introduced in [ICST].

Memory Management Unit

This is the security function SEF8 "Memory Management Unit (MMU)" introduced in [ICST].

Cryptographic Support

This is the security function SEF9 "Cryptographic Support" introduced in [ICST].

Application note:

This security function provides the basic ciphering block for all the DES cryptographic functions of the software layer.

8 Extended requirements

8.1. Extended families

8.1.1. Extended family FCS_RND - Generation of random numbers

8.1.1.1. Description

This family defines quality requirements for the generation of random numbers which are intended to be used for cryptographic purposes.

8.1.1.2. Rationale

This family has been introduced in [SSVG]. An attacker may predict or obtain information about random numbers generated by the TOE for instance because of a lack of entropy of the random numbers provided. Here the attacker is expected to take advantage of statistical properties of the random numbers generated by the TOE without specific knowledge about the TOE's generator. Malfunctions or premature aging are also considered which may assist in getting information about random numbers. To counter this kind of attacks, the TOE must ensure the cryptographic quality of random number generation. For instance random numbers shall not be predictable and shall have a sufficient entropy. The introduction of this new class enables to specify the quality metric that must be used.

8.2. Extended components

8.2.1. Extended component FCS_RND.1

8.2.1.1. Description

The generation of random numbers requires that random numbers meet a defined quality metric.

There are no management activities foreseen for this component.

There are no actions defined to be auditable for this component.

8.2.1.2. Definition

FCS_RND.1 Quality metric for random numbers

FCS_RND.1.1 The TSF shall provide a mechanism to generate random numbers that meet [assignment: a defined quality metric].

Dependencies: No dependencies

8.2.1.3. Rationale

It was chosen to define FCS_RND.1 explicitly, because Part 2 of the Common Criteria do not contain generic security functional requirements for Random Number generation. Note tha there are security functional requirements in Part 2 of the Common Criteria, which refer to random numbers. However, they define requirements only for the authentication context, which is only one of the possible applications of random numbers.

8.2.2. Extended component FPT_TST.2

8.2.2.1. Description

Subset TOE security testing, provides the ability to test the correct operation of particular security functions or mechanisms. These tests may be performed at start-up, periodically, at the request of the authorised user, or when other conditions are met. It also provides the ability to verify the integrity of TSF data and executable code.

8.2.2.2. Definition

FPT_TST.2 Subet TOE testing

FPT_TST.2.1 The TSF shall run a suite of self tests [selection: during initial start-up, periodically during normal operation, at the request of the authorized user, [assignment: conditions under which the self test should occur]] to demonstrate the correct operation of [assignment: functions and/or mechanism].

Dependencies: (FPT_AMT.1)

8.2.2.3. Rationale

The security is strongly dependent on the correct operation of the security functions. Therefore, the TOE shall support that particular security functions or mechanisms are tested in the operational phase (Phase 7). The tests can be initiated by the Smartcard Embedded Software and/or by the TOE. Part 2 of the Common Criteria provides the security functional component "TSF testing (FPT_TST.1)". The component FPT_TST.1 provides the ability to test the TSF?s correct operation. For the user it is important to know which security functions or mechanisms can be tested. The functional component FPT_TST.1 does not mandate to explicitly specify the security functions being tested. In addition, FPT_TST.1 requires verification of the integrity of TSF data and of the stored TSF executable code which might violate the security policy. Therefore, the security functional component Subset TOE security testing (FPT_TST.2) has been newly created. This component allows that particular parts of the security mechanisms and functions provided by the TOE are tested.

Index

A

A.APPLET ... 34
A.MANUFACTURING ... 34
A.NATIVE ... 33
A.VERIFICATION ... 33
Administration__Commands__Control ... 111
Application__Provider ... 24
Atomic__Transactions ... 114

B

Booting__Tests ... 114

C

Card User ... 25
Card__Administrator ... 24
Card__Content__Management ... 111
Card__Enabler ... 24
Card__Locking ... 113
Card__Manufacturer ... 24
Card__Muting ... 113
Card__Termination ... 113
Cardholder ... 24
Cardholder__Verification ... 116
Clearing__sensitive__information ... 114
Cryptographic__Support ... 117

D

D.API_DATA ... 31
D.APP_C_DATA ... 30
D.APP_I_DATA ... 30
D.APP_KEYs ... 30
D.APP-CODE ... 28
D.APP-INST ... 28
D.COMMAND ... 28
D.CRYPTO ... 31
D.CVM ... 29
D.GP-REGISTRY ... 29
D.ISD-PERSO ... 29
D.ISD-SESSION-KEYS ... 29
D.JCS_DATA ... 31
D.PIN ... 30
D.SEC_DATA ... 31
D.SOFTWARE ... 30
Defensive__Java__Card__Virtual__Machine ... 113

E

Encryption__and__Decryption ... 115

F

FAU_ARP.1-JCS ... 92
FCO_NRO.2-SC ... 67
FCS_CKM.1-APP-DH ... 90
FCS_CKM.1-APP-EC ... 89
FCS_CKM.1-APP-RSA ... 89
FCS_CKM.1-SCP-SESSION-KEYS ... 77
FCS_CKM.2-APP-DH-EC ... 90
FCS_CKM.2-APP-DH-EG ... 90
FCS_CKM.2-SCP-SESSION-KEYS ... 77
FCS_CKM.3-KL ... 76
FCS_CKM.4-KD ... 78
FCS_COP.1-APP-CIPHER/DES ... 88
FCS_COP.1-APP-RIPMD ... 88
FCS_COP.1-APP-RSA ... 89
FCS_COP.1-APP-SHA ... 88
FCS_COP.1-APP-SIGN/DES ... 88
FCS_COP.1-DAP ... 75
FCS_COP.1-IC ... 103
FCS_COP.1-SCP/FULL ... 75
FCS_COP.1-SCP02/CBC ... 74
FCS_COP.1-SCP02/ECB ... 74
FCS_COP.1-SCP02/FINAL ... 75
FCS_COP.1-SCP02/ICV ... 75
FCS_RND.1-APP ... 89
FDP_ACC.1-DVM ... 98
FDP_ACC.1-ISD ... 60
FDP_ACC.1-MMU ... 105
FDP_ACC.2-ADEL ... 96
FDP_ACC.2-FIREWALL ... 82
FDP_ACF.1-ADEL ... 96
FDP_ACF.1-DVM ... 98
FDP_ACF.1-FIREWALL ... 83
FDP_ACF.1-ISD ... 62
FDP_ACF.1-MMU ... 105
FDP_IFC.1-IC ... 104
FDP_IFC.1-JCVM ... 85
FDP_IFC.2-BCV ... 107
FDP_IFC.2-SCP ... 69
FDP_IFF.1-JCVM ... 85
FDP_IFF.1-SCP ... 70
FDP_IFF.2-BCV ... 108
FDP_ITC.1-KL ... 76
FDP_ITC.2-CCM ... 79
FDP_ITT.1-IC ... 104
FDP_RIP.1-ABORT ... 91
FDP_RIP.1-ADEL ... 99
FDP_RIP.1-APDU ... 91
FDP_RIP.1-bArray ... 91
FDP_RIP.1-KEYS ... 91
FDP_RIP.1-OBJECTS ... 86
FDP_RIP.1-ODEL ... 100
FDP_RIP.1-TRANSIENT ... 91
FDP_ROL.1-CCM ... 79
FDP_ROL.1-FIREWALL ... 92
FDP_SDI.2 ... 92
FDP_UIT.1-CCM ... 78
FIA_AFL.1-CVM ... 81
FIA_AFL.1-SC ... 74
FIA_ATD.1-AID ... 95
FIA_UAU.1-SC ... 68
FIA_UAU.4-SC ... 68

FIA_UID.1-SC 68
FIA_UID.2-AID 95
FIA_USB.1 95
FMT_MSA.1-BCV.1 108
FMT_MSA.1-BCV.2 109
FMT_MSA.1-ISD.1 64
FMT_MSA.1-ISD.2 64
FMT_MSA.1-ISD.3 64
FMT_MSA.1-ISD.4 64
FMT_MSA.1-JCRE 86
FMT_MSA.1-MMU 106
FMT_MSA.1-SCP-BOTH 72
FMT_MSA.1-SCP-OFFCARD 72
FMT_MSA.1-SCP-ONCARD 72
FMT_MSA.2-BCV 109
FMT_MSA.2-JCRE 87
FMT_MSA.2-KEYS 77
FMT_MSA.3-ADEL 97
FMT_MSA.3-BCV 109
FMT_MSA.3-DVM 99
FMT_MSA.3-FIREWALL 87
FMT_MSA.3-ISD 64
FMT_MSA.3-MMU 106
FMT_MSA.3-SCP 73
FMT_MTD.1-JCRE 94
FMT_MTD.3-AID 94
FMT_SMF.1-BCV 109
FMT_SMF.1-FIREWALL 87
FMT_SMF.1-ISD 65
FMT_SMF.1-MMU 106
FMT_SMF.1-SCP 73
FMT_SMR.1-ADEL 98
FMT_SMR.1-BCV 109
FMT_SMR.1-CA 73
FMT_SMR.1-ISD 65
FMT_SMR.1-JCRE 87
FMT_SMR.1-PRV 66
FPR_UNO.1-CRYPTO 93
FPR_UNO.1-PIN 94
FPT_AMT.1-OS 101
FPT_FLS.1-ADEL 100
FPT_FLS.1-CCM 80
FPT_FLS.1-IC 102
FPT_FLS.1-JCS 93
FPT_FLS.1-ODEL 100
FPT_ITT.1-IC 104
FPT_PHP.3-IC 103
FPT_RCV.3-CCM/AI 80
FPT_RCV.3-OS 101
FPT_RCV.4-OS 101
FPT_RVM.1 93
FPT_SEP.1-FIREWALL 87
FPT_SEP.1-IC 102
FPT_TDC.1 93
FPT_TDC.1-KL 76
FPT_TST.1 94
FPT_TST.2-IC 103
FRU_FLT.2-IC 102
FRU_RSA.1-BCV 109
FRU_RSA.1-CCM 81
FTP_ITC.1-SC 66

G

Global__CVM 112

H

Hardware__Data__Encryption 117
Hardware__Self__Test 117
Host__Authentication 112

I

IC__Manufacturer 24
ISD__Key__Loading__and__Replacement 112

J

Java__Card__Firewall 113

K

Key__Agreement 116
Key__Destruction 116
Key__Encryption 116
Key__Generation 115
Key__Integrity 116

L

Life__Cycle__Management 111

M

Manufacturer 23
Memory__Management__Unit 117
Message__Data__Confidentiality 112
Message__Digest__Generation 115
Message__Integrity__and__Authentication 112

N

Notification__of__Physical__Attack 117

O

O.ALARM 54
O.CIPHER 54
O.CVM-BLOCK 51
O.DELETION 50
O.ERROR-COUNTERS 51
O.FIREWALL 52
O.GLOBAL-CVM 51
O.IC-SUPPORT 56
O.IDENTIFICATION 50
O.INFO-CONFIDENTIALITY 49
O.INFO-INTEGRITY 48
O.INFO-ORIGIN 48
O.INSTALL 49
O.KEY-MNGT 55
O.LIFE-CYCLE 50
O.LOAD 49
O.LOCK 52

O.NATIVE 53
O.NO-KEY-REUSE 49
O.OBJ-DELETION 55
O.OPERATE 52
O.OS-SUPPORT 54
O.PIN-MNGT 54
O.PROT-INF-LEAK 56
O.PROT-MALFUNCTION 57
O.PROT-PHYS-TAMPER 56
O.REALLOCATION 53
O.RECOVERY 51
O.REQUEST 48
O.RESOURCES 52
O.RND 57
O.SHRD_VAR_CONFID 53
O.SHRD_VAR_INTEG 53
O.SID 51
O.TRANSACTION 54
O.VERIFICATION 53
OE.APPLET 58
OE.KEY-LENGTH 58
OE.MANUFACTURING 59
OE.NATIVE 58
OE.NO-RMI-APPLETS 59
OE.SECRETS 58
OE.VERIFICATION 58
OPEN 111
Operating__State__Checking 116
OSP.FILE-ORIGIN 46
OSP.KEY-LENGTH 46
OSP.NO-RMI-APPLETS 46
OSP.PERSONALIZATION 47
OSP.PROCESS-TOE 47
OSP.SECRETS 46
OSP.VERIFICATION 45

P

Phase__Management 116
PIN__Value__Integrity 116
Platform__Developer 24
Protection__Against__Snooping 117

R

Random__Number__Generation 115
Resource__Quotas 114

S

S.APP 32
S.BCV 33
S.ISD 32
S.JCRE 33
S.NAT 32
S.OPEN 32
S.SPY 32
Secure__Channel__Termination 112
Session__Key__Generation 112
Signature__Generation__and__Verification 114

T

T.BRUTE-FORCE 38
T.CONFID-APPLI-DATA 40
T.CONFID-JCS-CODE 39
T.CONFID-JCS-DATA 40
T.CRYPTO 43
T.DELETION 39
T.EXE-CODE.1 42
T.EXE-CODE.2 42
T.FORCED-RESET 37
T.IMPERSONATE 35
T.INF-LEAK 44
T.INSTALL 38
T.INTEG-APPLI-CODE.1 40
T.INTEG-APPLI-CODE.2 41
T.INTEG-APPLI-DATA.1 40
T.INTEG-APPLI-DATA.2 41
T.INTEG-JCS-CODE 40
T.INTEG-JCS-DATA 40
T.INVALID-INPUT 35
T.INVALID-ORDER 36
T.LIFE-CYCLE 38
T.MALFUNCTION 44
T.NATIVE 42
T.OBJ-DELETION 43
T.PHYS-TAMPER 44
T.REPLAY 37
T.RESSOURCES 43
T.RND 45
T.SID.1 41
T.SID.2 42
True__Random__Number__Generation 117

V

Verification Authority 24