COSMO V 7.1-S
TOUTATIS

JAVA CARD OPEN PLATFORM

PUBLIC SECURITY TARGET

Table of contents

List of figures

List of tables

1 Security Target Introduction

This Security Target aims to satisfy the requirements of Common Criteria level EAL5+, augmented with AVA_VAN.5 and ALC_DVS.2 in defining the security enforcing functions of the Target Of Evaluation and describing the environment in which it operates.

The basis for this composite evaluation is the composite evaluation of Platform and the hardware plus the cryptographic library.

1.1 Security Target reference

The following table defines the information related to the security target and associated evaluation.

Title:	TOUTATIS Security Target for ▪ ST23YR80/48 ▪ ST23YL80
Product Name:	Cosmo v 7.1-s
Editor:	Oberthur Technologies
OT registration:	FQR 110 6155
EAL:	EAL5+, augmented with: ▪ ALC_DVS.2 ▪ AVA_VAN.5
ITSEF:	CEA LETI
Certification Body:	ANSSI
Evaluation scheme:	French

More precisely, the security target describes:

- The Target Of Evaluation (TOE), including the TOE components, the components in the TOE environment, the product type and its life cycle
- The TOE security environment TOE, including assets to be protected and threats to be countered by the TOE and by the operational environment during the development and the platform active phases
- The TOE security objectives and its supporting environment in terms of integrity and confidentiality of sensitive information of the TOE
- The organizational security policies and the assumptions
- The security requirements which include the TOE functional requirements, the TOE assurance requirements and the security requirements for the environment
- The summary of the TOE specification including a description of the security functions and assurance measures that meet the TOE security requirements

This V7.1-s platform is able to receive and manage different types of applications (IAS, LDS and ID-One Classic …).

Some of these applications are in ROM (already loaded in the platform), others can be loaded in EEPROM at the Personalisation phase or at the use phase.

1.2 TOE Reference

TOE Name	**ID-One Cosmo V7.1-s Standard Dual**	**ID-One Cosmo V7.1-s Basic Dual**	**ID-One Cosmo V7.1-s Standard**
Mask / Hardware Identification	61 01 20	61 01 04	61 01 0C
Label PVCS code	V71ST V1.10	V71ST V1.10	V71ST V1.10
Optional Code Generic on ID-One Cosmo v7.1-s Identification (mandatory)	078624	078912	079722
IC reference version	ST23YR80B	ST23YR48B	ST23YL80C
IC ST identification	ST23YR48B/ST23YR80B Security Target EAL6 augmented by ALC_FLR.1	ST23YR48B/ST23YR80B Security Target EAL6 augmented by ALC_FLR.1	ST23YL80C Security Target EAL5 augmented by ALC_DVS.2 and AVA_VAN.5
IC certificate	ANSSI-CC-2010/01	ANSSI-CC-2010/01	ANSSI-CC-2009/37
Optional Code SAC on ID-One Cosmo v7.1-s Identification (optional)	079212	▨	079212

2 TOE overview

The Smart Card intended to support the TOE is composed of hardware and software components, as listed below and described in Figure 1.

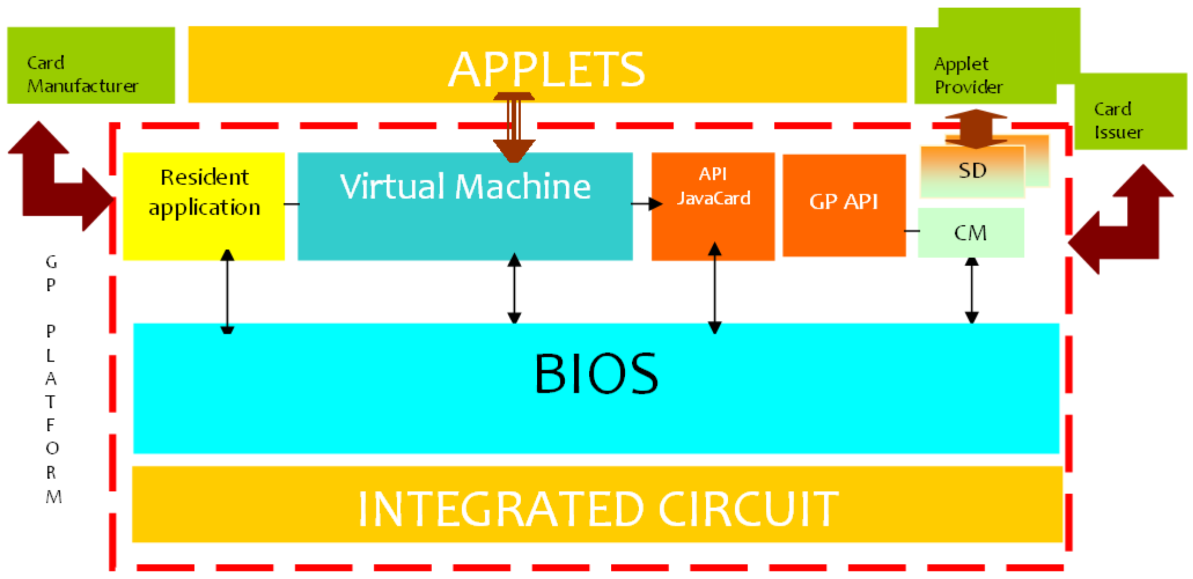


Figure 1: (Java Platform Architecture)

The TOE includes the BIOS, the Virtual Machine, the APIs, the Global Platform application, the Resident application and the IC component. Details of components are presented in the TOE description.

2.1 TOE Type

ID-One Cosmo V7.1-s on ST23 family is a contact/dual/contactless javacard platform based, compatible with multi-application ID-One Cosmo product family.

The functional level of the OS will be based on a Java™ based multi-application platform, compliant with **Java Card 3.0.1 classic edition** and **Global Platform 2.2.1** specifications.

2.1.1 Java Card Platform

The Java technology, embedded on the TOE, combines a subset of the Java programming language with a runtime environment optimized for smart cards and similar small-memory embedded devices.

The Java Card^{TM} platform is a smart card platform enabled with Java Card^{TM} 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 3.0.1 classic edition, specified by three documents related to Java Card API, Java Card Runtime Environment and Java Card Virtual Machine Specifications, defined respectively in **[R6]**, **[R7]** and **[R8]**. The next paragraph introduces those three elements.

As the terminology is sometimes confusing, the term “Java Card System” has been introduced in **[R5]** that defines the set constituted by the Java Card RE, the Java Card VM and the Java Card API.
The Java Card System 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 Java Card VM is a bytecode interpreter embedded in the smart card. The Java Card RE is responsible for card resource management, communication, applet execution, on-card system and applet security.

Applet isolation is achieved through the Java Card Firewall mechanism defined in **[R7]**. This mechanism confines an applet to its own designated memory area. Thus, each applet is prevented from accessing fields and operations related to objects owned by other applets, unless those applets provide a specific interface (shareable interface) for that purpose. This access control policy is enforced at runtime by the Java Card VM.
However, applet isolation cannot be entirely granted by the firewall mechanism if certain well-formedness conditions are not satisfied by loaded applications.

Therefore, a bytecode verifier (BCV) formally verifies those conditions. The BCV is out of the scope of the Java Card System defined in **[R5]**.

The following Bytecode verification exists:

Platform Configuration	**Verifier**	**Type**	**When?**
Classic	Off-card bytecode verifier	Static	Once, outside of the card
Defensive	On-card bytecode verifier	Static	Once, on the card during the loading
Oberthur Technologies	Runtime verifier	Dynamic	Every time, during execution

The Java Card API (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 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 ISO/IEC 7816 and industry specific standards.

2.1.2 Global Platform

The TOE is compliant with the Global Platform 2.2.1 (GP) standard **[R9]** which provides a set of APIs and technologies to perform in a secure way, the operations involved in the management of the applications hosted by the card. Using GP maximizes the compatibility and the opportunities of communication as it becomes the current card management standard.

The main features addressed by GP are:

- The authentication of users through secure channels
- The downloading, installation removal, and selection for execution of Java Card applications
- The life cycle management of both the card and the applications
- The sharing of a global common PIN among all the applications installed on the card

These operations are addressed by a set of APIs used by the applications hosted on the card in order to communicate with the external world on a standard basis.

The version considered in this document is version 2.2.1 of the GP Card specification. The following GP functionalities, at least, are present within the TOE:

- Card content loading
- Extradition
- Asymetric keys
- DAP support, Mandated DAP support
- DAP calculation with asymmetric cryptography
- Logical channels
- SCP02 support
- SCP03 support **[R12]**
- Support for contact and contactless cards different implicit selection on different interfaces and channels
- Support for Supplementary Security Domains
- Trusted path privileges
- Post-issuance personalisation of Security Domain **[R12]**
- Application personalisation **[R12]**

2.1.3 Integrated Circuit (IC)

2.1.3.1 ST23YR48/80

The IC is a STMicroelectronics dual interface component that supports ISO/IEC 14443 Type B.

It is a hardware device composed of a processing unit, memories, security components and I/O interfaces. It has to implement security features able to ensure:

- The confidentiality and the integrity of information processed and flowing through the device,
- The resistance of the security IC to externals attacks such as physical tampering, environmental stress or any other attacks that could compromise the sensitive assets stored or flowing through it.

More information regarding the documentation is available in the public security target of the chip **[R26]**.

2.1.3.2 ST23YL80

The IC is a STMicroelectronics pure contact interface component.

It is a hardware device composed of a processing unit, memories, security components and I/O interfaces. It has to implement security features able to ensure:

- The confidentiality and the integrity of information processed and flowing through the device,
- The resistance of the security IC to externals attacks such as physical tampering, environmental stress or any other attacks that could compromise the sensitive assets stored or flowing through it.

More information regarding the documentation is available in the public security target of the chip **[R25]**.

2.1.4 Operating System (OS)

The TOE relies on an Operating System (OS) which is an embedded piece of software loaded into the Security IC. The Operating System manages the features and resources provided by the underneath chip. It is, generally divided into two levels:

1. Low level:
    a. Drivers related to the I/O, RAM, ROM, EEPROM, , and any other hardware component present on the Security IC
2. High level:
    a. Protocols and handlers to manage I/O
    b. Memory and file manager
    c. Cryptographic services and any other high level services provided by the OS

2.1.4.1 BIOS

The BIOS is an interface between hardware and native components like VM and APIs. The BIOS implements the following functionalities:

- APDU management, using T=0, T=1 and T=CL protocols
- Timer management
- Exceptions management
- Transaction management
- EEPROM access

TOUTATIS on ST23YR48/80:

Interfaces for the ST23YR48/80	
Contactless	References are standard ones
Contact	*
Dual	*

TOUTATIS on ST23YL80:

Interface for the ST23YL80	
Contact	References are standard ones

2.1.4.2 Cryptographic features

The following crypto services are included in the OS:

Cryptographic Services	
RSA CRT from 64 to 2048-bits by step of 32-bits	References are standard ones
RSA SFM from 64 to 2048-bits by step of 32-bits	*
ECC with 160, 192, 256, 384, 512 and 521-bits key sizes	*
TDES with 56, 112 and 168-bits key sizes	*
AES with 128, 192, 256 key sizes	*
SHA-1, SHA 224, 256, 384 and 512	*
RSA, ECC Key generation	*
CRC 16 and 32	*
RNG FIPS DES SP800-90	*
RSA signature/verification	Based on supported RSA key sizes
ECDSA signature/verification	Based on supported ECC key sizes
ECDH	Based on supported ECC key sizes
AES secure messaging	*
TDES secure messaging	*

2.1.4.3 Biometric feature

TOUTATIS embeds the MOC algorithm.
The biometric feature allows matching a CANDIDATE Template with REFERENCE Templates (up to 10)

2.1.4.4 Virtual Machine

The Virtual Machine, which is compliant with the Java Card 3.0.1 classic edition, interprets the byte code of Java Card applets.

The Virtual Machine supports logical channels; this means that it allows an applet to be selected on a channel, while a different applet is selected on another channel.

It also supports secure execution of applets loaded and stored in ROM.

The Virtual Machine is activated upon the selection of an applet.

2.1.4.5 The Java Card Runtime Environment

The Java Card Runtime Environment (JCRE) contains the Java Card Virtual Machine (VM), the Java Card Application Programming Interface (API) classes and industry-specific extensions, and support services. For details please refer to reference **[R7]**.

2.1.4.6 APIs

The APIs, compliant with the Java Card 3.0.1 classic edition, support key generation, Key Agreement, signature, ciphering of messages and proprietary OT API.

Proprietary APIs have been developed like ISOSecureMessaging to assure the data are exchanged in confidentiality and integrity; OTPinBio to compare a candidate fingerprint template with one of the reference fingerprint template previously store in the card; utilBER_Reader to read BER-TLV; SecureStore to store integrity sensitive information

2.1.4.7 Open and isolating Platform

This security target claims conformance to the Application Note 10 on Open and Isolating platform, issued by ANSSI **[R29]**.

An “open platform” can host new applications:

- Before its delivery to the end user (during phases 4, 5 or 6 of the traditional smartcard lifecycle). Such loadings are called “pre-issuance”.
- After its delivery to the end user (phase 7). Such loadings are called “post-issuance”.

An “isolating platform” is a platform that maintains the separation of the execution domains of all embedded applications on a platform, as of the platform itself. “Isolation” refers here to domain separation of applications as well as protection of application’s data.

2.1.4.8 Resident Application

It provides a native code application, with a basic main dispatcher, to receive the card commands and dispatch them to the application and module functions to implement the application commands.

It also deals with the Card Manufacturer authentication and logical channels management.

The dispatcher is always activated. Some card commands (for administration) are only available during prepersonalisation phase.

2.1.4.9 Applets

TOUTATIS platform embeds applets on the ROM. In conformance with ANSSI Note 10 **[R41]**, the applets have been provided to the ITSEF.

The applets list is provided in **[R32]**.

2.2 Major Security feature of the TOE

The main goal of the TOE is to provide a sound and secure execution environment to critical assets that need to be protected against unauthorized disclosure and/or modification.

The TOE with its security function has to protect itself and protect applets from bypassing, abuse or tampering of its services that could compromise the security of all sensitive data. Even if the applets are not in the scope of this evaluation.

Atomic Transactions

The TOE shall provide a transaction mechanism. It shall execute a sequence of modifications and allocations on the persistent memory so that either all of them are completed, or the TOE behaves as if none of them had been attempted.

The transaction mechanism shall permit to update internal TSF data as well as to perform different functions of the TOE, like installing a new package on the card.

This mechanism shall be available for applet instances

The TOE shall perform the necessary actions to roll back to a safe state upon interruption.

Card Content Management

The TOE shall control the loading, installation, and deletion of packages and applet instances.

To remove the code of a package from the card, or to definitely deactivate an applet instance, so that it becomes no longer selectable; it shall perform physical removal of those packages and applet data stored in memories (except applet in ROM memory that shall only be logically removed).

Card Management Environment

This function shall initialize and manage the internal data structure of the Card Manager. During the initialization phase of the card, it creates the Installer and the Applet Deletion Manager and initializes their internal data structures. The internal data structure of the Card Manager includes the Package and Applet Registries, which respectively contains the currently loaded packages and the currently installed applet instances, together with their associated AIDs.

This function shall also be in charge of dispatching the APDU commands to the applet instances installed on the card and keeping trace of the currently active ones.

It therefore handles sensitive TSF data of other security functions, like the Firewall or the Remote Access Control function.

Cardholder Verification

The TOE shall implement mechanisms to identify and authenticate the user of the product. This function is available to applet instances.

Clearing of sensitive information

The TOE shall ensure that no residual information is available from memories, and shall protect sensitive information that is no longer used. The Platform has to securely clear and destroy this information. It concerns PINs, keys, sensitive data (such as BIOMETRIC_DATA), buffer APDU.

This function is also available to applet.

DAP Verification

An Application Provider may require that its Application code to be loaded on the card shall be checked for integrity and authenticity. The DAP Verification privilege of the Application Provider's Security Domain shall provide this service on behalf of the Application Provider. A Controlling Authority may require that all Application code to be loaded onto the card shall be checked for integrity and authenticity. The Mandated DAP Verification privilege of the Controlling Authority's Security Domain shall provide this service on behalf of the Controlling Authority.

Data coherency

As coherency of data should be maintained, and as power is provided by the CAD and might be stopped at all moment (by tearing or attacks), a transaction mechanism need to be implemented.

When updating data, before writing the new ones, the old ones are saved in a specific memory area. If a failure appears, at the next start-up, if old data are valid in the transaction area, the system restores them for staying in a coherent state.

Data integrity

Sensitive data have to be protected from modifications: keys, pins, patch code and sensitive applet data.

Encryption and Decryption

The TOE provides the applet instances with a mechanism for encrypting and decrypting the contents of a byte array.

Ciphering operations are implemented to resist environmental stress and glitches and include measures for preventing information leakage through covert channels.

Entity authentication/secure Channel

Off-card entity authentication is achieved through the process of initiating a Secure Channel and provides assurance to the card that it is communicating with an authenticated off-card entity.

If any step in the off-card authentication process fails, the process shall be restarted (i.e. new session keys generated).

The Secure Channel initiation and off-card entity authentication implies the creation of session keys derived from card static key(s).

Exception

In case of abnormal event: data unavailable on an allocation or illegal access to a data, the system shall own an internal mechanism allowing it to stop the code execution and raise an exception.

Firewall

The TOE with the Firewall shall control information flow at runtime. It shall ensure controls object sharing between different applet instances, and between applet instances and the Java Card RE.

GP_Dispatcher

While a Security Domain or Card Manager is selected, the TOE shall test for every command if Security Domain Owner authentication is required. If a secure channel is opened, the TOE tests according to the Security Domain state and the Card state for every command if secure messaging is required.

Hardware operating

The TOE shall boot after the IC has successfully powered-up. The TOE boot operations shall ensure the correct initialization of the TOE functionalities and the integrity of the code and data.

The TOE shall monitor IC detectors (e.g. out-of-range voltage, temperature, frequency, active shield, memory aging) and shall provide automatic answers to potential security violations through interruption routines that leave the device in a secure state.

Key Access

The TOE shall enforce secure access to all cryptographic keys on the card: RSA keys, DES keys, EC keys, AES keys

Key Agreement

The TOE shall provide to applet instances a mechanism for supporting key agreement algorithms such as EC Diffie-Hellman.

Key destruction

The TOE shall provide secure key destruction, such as keys can not be retrievec from erased data.

Key Distribution

The TOE shall enforce the distribution of all the cryptographic keys of the card using a specific method.

Key Generation

The TOE shall enforce the creation and the on card generation of all the cryptographic keys of the card using a specific method.

Key management

The TOE shall manage key set: Loading keys, adding a new key set (version and value of the key) or updating a key set (update key value).

Manufacturer Authentication

During prepersonalisation phase, manufacturer authentication at the beginning of a communication session shall be mandatory prior to any relevant data being transferred to the TOE.

Memory failure

This security functionality is in charge of the management of bad usage of the memory.

Message Digest

Message digest generation shall be implemented to resist environmental stress and glitches and include measures for preventing information leakage through covert channels.

The TOE shall provide the applet instances with a mechanism for generating an (almost) unique value for the contents of a byte array. This value can be used as a short representative of the information contained in the whole byte array.

For Hashing algorithms that do not pad the messages, the TSF checks that the information is block aligned before computing its hash value.

Pre-personalisation

This function shall permit to pre-initialize the internal data structures, to load the configuration of the card and to load patch code if needed and locks.

The TOE shall allow loading of TOE sensitive data: configuration data. Configuration data can contain patchs. The TOE shall check the integrity of the incoming data. Unless stated otherwise, the origin of the incoming data shall be ensured by organisational means. The TOE shall ensure that TOE code and patchs installed after delivery cannot be bypassed. The loading functionality of patchs shall be disabled before entering the final usage phase. The TOE identification shall take into account the patchs installed after delivery.

Random Number

This TOE functionality provides the card manager, the resident application and the applets a mechanism for generating challenges and key values.

The Number Generator is a combination of hardware and software RNG. The RNG is compliant with **[R30]**.

Resident Application dispatcher

During prepersonalisation phase, this function shall verify for every command if manufacturer authentication is required.

Remote access

During prepersonalisation phase, this function shall verify for every command if manufacturer authentication is required.

Runtime Verifier

This security functionality ensures the secure processing of the stack, heap and transient by ensuring additional controls.

Security functions of the IC

This TOE functionality ensures the correct execution of the IC functionalities.

Signature

This TSF shall provide the applet instances with a mechanism for generating an electronic signature of the contents of a byte array and verifying an electronic signature contained in a byte array.

An electronic signature is made of a hash value of the information to be signed, encrypted with a secret key. The verification of the electronic signature includes decrypting the hash value and checking that it actually corresponds to the block of signed bytes. Signature operations shall be implemented to resist environmental stress and glitches and include measures for preventing information leakage through covert channels.

Unobservability

The TOE shall use and manipulate sensitive information without revealing any element of this information.

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

The only non-TOE component required on the product is the bytecode verifier. The bytecode verifier is a program that performs static checks on the bytecodes of the methods of a CAP file.

Bytecode verification is a **key** component of security: applet isolation, for instance, depends on the file satisfying the properties a verifier checks to hold. A method of a CAP file that has been verified shall not contain, for instance, an instruction that allows forging a memory address or an instruction that makes improper use of a return address as if it were an object reference. In other words, bytecodes are verified to hold up to the intended use to which they are defined. This TOE considers static bytecode verification; it has to be performed on the host at off-card verification and prior to the installation of the file on the card in any case.

2.4 TOE usage

This Platform is an open and isolating platform that is compliant with the ANSSI Application Note 10 that deals with open and isolating platforms.

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

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

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

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

So far, the most typical applications are:

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

- Secure information storage, like health records, or health insurance cards.
- Loyalty programs, like the “Frequent Flyer” points awarded by airlines. Points are added and deleted from the card memory in accordance with program rules. The total value of these points may be quite high and they must be protected against improper alteration in the same way that currency value is protected.

For more information on embedded applets and known applets, refer to §2.1.4.9.

This platform must provide a highly secure technology for smartcards applications. The Match-On-Card technology is an entire part of the product, and enables the Authentication by way of digital prints.

This secure platform is compliant with the security requirements (RGS_B1, RGS_B2 and RGS_B3) of *“Référentiel général de la sécurité des systèmes d’informations”* **[R31]**, with the maximum level of trust: *“qualification renforcée”.*

This document is edited by ANSSI and is a requirement for highly secure product, destined to Government market, such as Passport and ID cards.

2.5 TOE Guidances

The ID-One Cosmo V7.1-s is evaluated with its guidance. The guidances of the Platform are listed hereafter:

Guide	Ref	Title
[GUIDE1]	**[R37]**	ID-One Cosmo V7.1 Security Recommendations FQR 110 6029
[GUIDE2]	**[R38]**	ID-One Cosmo V7.1 Reference Guide FQR 110 6028
[GUIDE3]	**[R39]**	ID-One Cosmo V7.1 Pre-Perso Guide FQR 110 6027
[GUIDE4]	**[R40]**	ID-One Cosmo V7.1 Application Loading Protection Guidance FQR 110 6267
[GUIDE5]	**[R32]**	Applications on ID ONE COSMO V7.1-S FQR 110 6268

The platform is evaluated without applications.
Applications need to be verified by the Verifier before being loaded.

[GUIDE1]
If the applet needs to have a security certification, the applet must follow recommendations listed in the document.
If the applet does not need additional security certification with the platform, the certificate of the Platform is still valid if the applet go through the verifier when this applet is loaded (the security function of the platform are still ok).
The [GUIDE1] is provided to the Developer of an application to be certified.

[GUIDE2]

This document describes the ID-One Cosmo V7.1-s smart card usage. It describes how to use the card from an APDU commands point of view and gets onto topics such as common platform APDU commands, secure channels and security domains.
This document also describes the available javacard and proprietary APIs for applet developers.
The [GUIDE2] is provided to the Developer of an application to be certified or not, and also to the final user (in phases 6-7).

[GUIDE3]
This document describes the pre-personalisation steps that should be followed to correctly initialize the Cosmo v7.1 platforms. The TOE is finalized once it’s prepersonalised.
This document is provided to the final user (phases 4-5).

[GUIDE4]
This document describes the loading procedure, in compliance with ANSSI Note 10 and the Java Card Open Platform protection profile.
The [GUIDE4] is provided to the Loading Authority, who is in charge of loading an application.

[GUIDE5]
This document identifies the known applications and the one that could be embedded in the ID-One Cosmo v7.1-s.
The [GUIDE5] is provided to the Application Provider, the loading authority and the final user. (phase 4,5, 6 and 7).

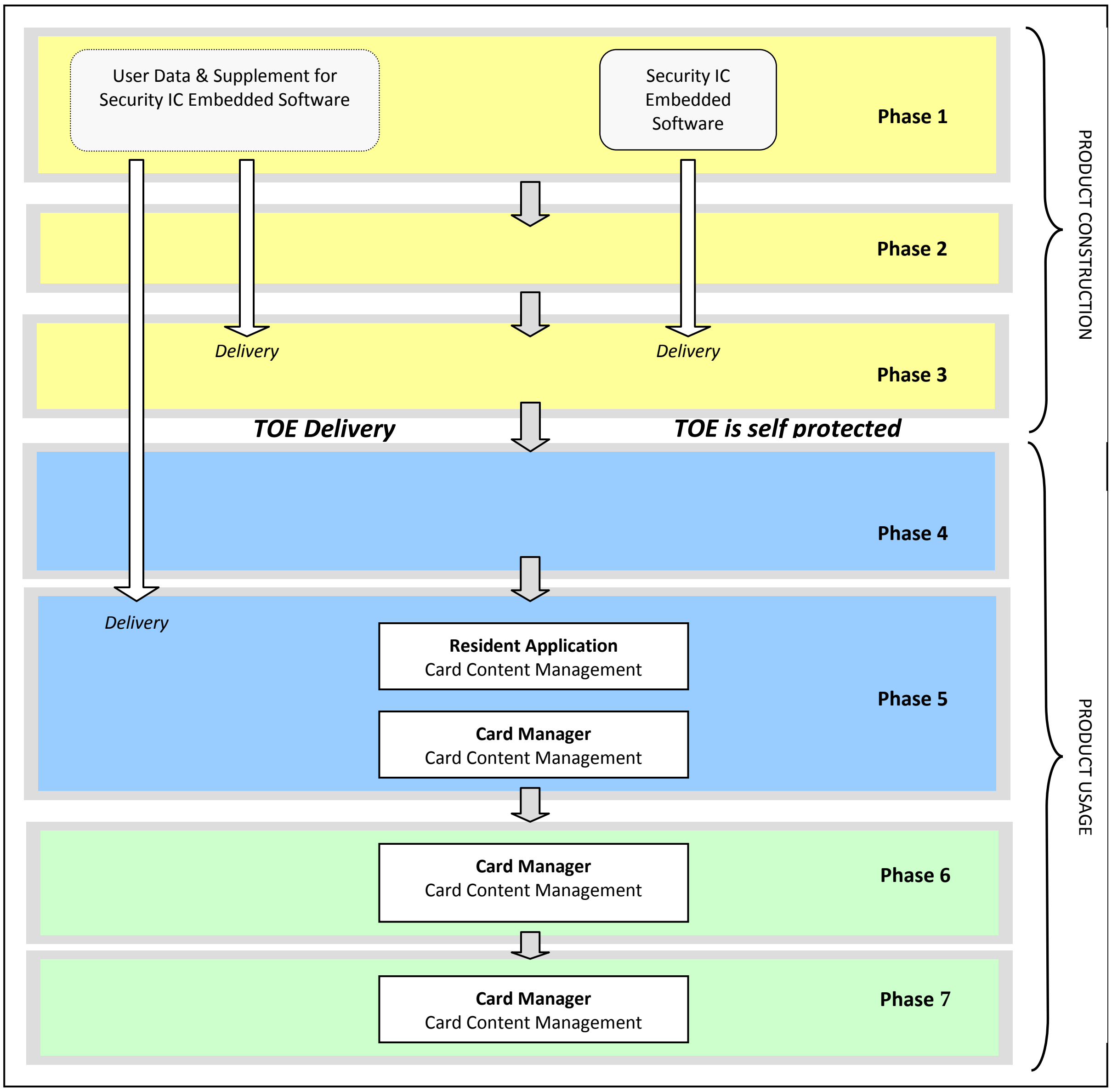
2.6 TOE Life cycle

The development and manufacturing processes of the Composite Product is separated into seven distinct phases to be in accordance with the Java Card™ System Protection Profile (section 3.2). Each phase is under the control of one (or several) administrator(s) and protected by an environment. Each phase is covered by the assurance components.

Please note that AGD_PRE stands for [GUIDE3] and AGD_OPE stands for [GUIDE4].

Phase	Phase name	Covered by
1	Security IC Embedded Software development	ALC [TOUTATIS]
2	Security IC Development	ALC [IC]
3	Security IC Manufacturing	ALC [IC]
4	Security IC Packaging	AGD_PRE [TOUTATIS]
5	Composite Product Integration	AGD_PRE [TOUTATIS]
6	Composite Product Personalisation	AGD_OPE [TOUTATIS]
7	Operational Usage	AGD_OPE [TOUTATIS]

Table 1: (TOE Life Cycle – Summary)



User Data & Supplement for Security IC Embedded Software
Security IC Embedded Software
Phase 1
Phase 2
Delivery
Delivery
Phase 3
TOE Delivery
TOE is self protected
PRODUCT CONSTRUCTION
Phase 4
Delivery
Resident Application Card Content Management
Card Manager Card Content Management
Phase 5
Card Manager Card Content Management
Phase 6
Card Manager Card Content Management
Phase 7
PRODUCT USAGE

Figure 2: (TOE Life Cycle – Overviews)

2.7 Software Components Life Cycle

2.7.1 Card Life Cycle

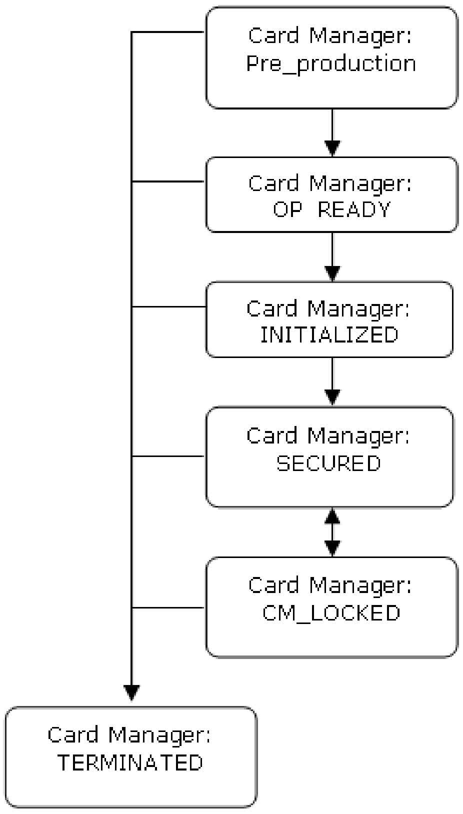


Figure 3: (Card Life Cycle)

2.7.1.1 Pre_production

This initial life state of the Card allows managing the prepersonalisation of the Javacard Platform Embedded Software up to the Card Manager Life Cycle OP_READY.
During this state, the Resident Application provides a set of APDU commands which allows:

- Writing User Data for configuring the Javacard Platform Embedded Software. This configuration (by using lock mechanism) is only carried out during this state.
- Writing Supplement for Javacard Platform Embedded Software (patch code). It is developed at Oberthur Technologies premises (phase 1), delivered and loaded securely in volatile memory (EEPROM) during the Composite Product Integration (phase 5). The security of this loading is fully enforced by technical measures provided by the TOE, and evaluated by the ITSEF. This task is only carried out during this state.
- Activating Load Files from immutable persistent memory (ROM). This task is only carried out during this state.
- Loading Load Files from mutable persistent memory (EEPROM).
- Instantiating the Issuer Security Domain (Card Manager). Only one ISD is available by card.
- Populating with initialization key (ISK) and Chip CPLC. The Card Life Cycle switchs automatically in OP_READY state when the initialization key (ISK) is populated in the ISD.

The APDU commands depend on the mutual authentication carries out (by using MSK).

The next states possible are OP_READY or TERMINATED. The transition is irreversible.

2.7.1.2 OP_READY

During this life cycle state, all the basic functionalities of the runtime environment are available and the Card Manager is ready to receive, execute and respond to APDU commands. During this state, a new keyset have to be loaded before switching to INITIALIZED life state.
The card is assumed to have the following functionalites in the OP_READY state:

- The runtime environment is ready for execution.
- An Initialization key is available within the Card Manager.
- Card Content Management operations are supported.
- Post-issuance personalisation of applets belonging to the Card Issuer can be carried out via the Card Manager.

The next states possible are INITIALIZED or TERMINATED. The transition is irreversible.

2.7.1.3 INITIALIZED

This life state is an administrative card production state. Most of the personalisation of the Card Manager is performed when entering in this state.

The card is assumed to have the following functionalites in the INITIALIZED state:

- The runtime environment is ready for execution.
- A keyset is available within the Card Manager.
- Card Content Management operations are supported.
- Post-issuance personalisation of applets belonging to the Card Issuer can be carried out via the Card Manager.

The next states possible are SECURED or TERMINATED. The transition is irreversible.

2.7.1.4 SECURED

The Card life cycle state SECURED is the normal operating life cycle state of the card after issuance. This state is the indicator for the Card Manager to enforce the Card Issuer's security policies related to post-issuance card behaviour such as applet loading and activation.

The card is assumed to have the following functionality in the state SECURED:

- The Card Manager contains all necessary key sets and security elements for full functionality.
- Card Issuer initiated card content changes can be carried out through the Card Manager.
- Card Content Management operations are supported.
- Post-issuance personalisation of applets belonging to the Card Issuer can be carried out via the Card Manager.

The next states possible are CM_LOCKED or TERMINATED.

The transition in the TERMINATED state is irreversible.

2.7.1.5 CM_LOCKED

The state CM_LOCKED is used to instruct the Card Manager to temporarily disable all applets on the card except for the Card Manager. This state is created to give the Card Issuer the ability to temporarily disable functionality of the card on detection of security threats (either internal or external to the card).

Setting the Card Manager to this state implies that the card will no longer work, except via the Card Manager which is controlled by the Card Issuer. No Card Content Management operation is possible.

The next states possible are SECURED or TERMINATED.

The transition in the TERMINATED state is irreversible.

2.7.1.6 TERMINATED

The Card Manager is set to the life cycle state TERMINATED to permanently disable all card functionalities including the functionality of the Card Manager itself. This state is created as a mechanism for the Card Issuer to logically ‘destroy’ the card for such reasons as the detection of a severe security threat or upon expiration of the card.

Only GET DATA (CPLC) command is available. No Card Content Management operation is possible.

The Card Manager state TERMINATED is irreversible and signals the end of the card’s life cycle.

2.7.2 Security Domain Life Cycle States

The Security Domain Life Cycle begins when a Security Domain is instantiated in the card. The Security Domain Life Cycle States defined by Global Platform are INSTALLED, SELECTABLE, PERSONALIZED and LOCKED. There are no proprietary Security Domain Life Cycle States.

Figure 4: (Security Domain Life Cycle illustrates the state transition diagram for the Security Domain Life Cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

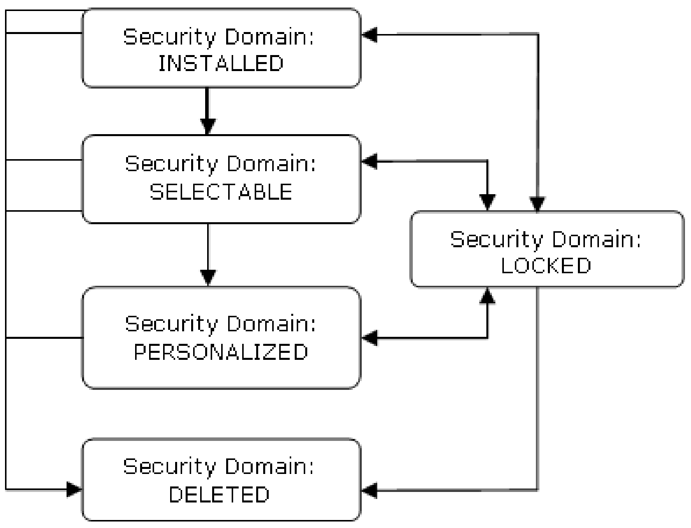

Security Domain: INSTALLED
Security Domain: SELECTABLE
Security Domain: LOCKED
Security Domain: PERSONALIZED
Security Domain: DELETED

Figure 4: (Security Domain Life Cycle)

2.7.2.1.1 INSTALLED

The state INSTALLED means that the Security Domain becomes an entry in the Global Platform Registry and this entry is accessible to off-card entities authenticated by the associated Security Domain. The Security Domain is not yet available for selection. It cannot be associated with Executable Load Files or Applications yet and therefore its Security Domain services are not available to Applications.

2.7.2.1.2 SELECTABLE

The state SELECTABLE means that the Security Domain is able to receive commands (specifically personalisation commands) from off-card entities. As they still do not have keys, the Security Domains cannot be associated with Executable Load Files or Applications and therefore their services are not available to Applications when they are in this state. The state transition from INSTALLED to SELECTABLE is irreversible. The transition to SELECTABLE may be combined with the Security Domain installation process.

2.7.2.1.3 PERSONALIZED

The definition of what is required for a Security Domain to transition to the state PERSONALIZED is Security Domain dependent but is intended to indicate that the Security Domain has all the necessary personalisation data and keys for full runtime functionality (i.e. usable in its intended environment). The transition from SELECTABLE to PERSONALIZED is irreversible.

In the state PERSONALIZED, the Security Domain may be associated with Applications and its services become available to these associated Applications.

2.7.2.1.4 LOCKED

The OPEN, the Security Domain itself, the Security Domain's associated Security Domain (if any), an Application with the Global Lock privilege or a Security Domain with the Global Lock privilege uses the state LOCKED as a security management control to prevent the selection of the Security Domain.

If the OPEN detects a threat from within the card and determines that the threat is associated with a particular Security Domain, that Security Domain may be prevented from further selection by the OPEN setting the Security Domain's Life Cycle State to LOCKED.

Alternatively, the off-card entity may determine that a particular Security Domain on the card needs to be locked for a business or security reason and may initiate the state transition via the OPEN.

Locking a Security Domain prevents this Security Domain from being associated with new Executable Load Files or Applications. In this state DAP verification, extradition and access to that Security Domain’s services shall fail. In summary, if a Security Domain is in the lifecycle state LOCKED, it shall reject all received commands.

Once the Life Cycle State is LOCKED, only the Security Domain's associated Security Domain (if any), an Application with Global Lock privilege or a Security Domain with Global Lock privilege is allowed to unlock the Security Domain. The OPEN shall ensure that the Security Domain's Life Cycle returns to its previous state.

2.7.2.1.5 DELETED

At any point in the Security Domain Life Cycle, the OPEN may receive a request to delete a Security Domain.

The space previously used to store a physically deleted Security Domain is reclaimed and may be reused. The entry within the Global Platform Registry shall no longer be available, and the OPEN is not required to maintain a record of the deleted Security Domain's previous existence.

2.7.3 Load File Life Cycle

The Load Files Life Cycle begins when a Load File is activated from immutable persistent memory (ROM) or loaded in mutable persistent memory (EEPROM).

Figure 5: (Load File Life Cycle) illustrates the state transition diagram for the Load File Life Cycle. This can typically be viewed as a sequential process.

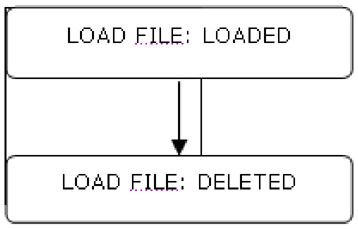


Figure 5: (Load File Life Cycle)

The Load Files activated (Phase 5) or loaded (Phase 5 and/or 6) must satisfy a process using the following tools:

- Compiler: software that generates machine-independent code (bytecode)
- Converter: software that preprocesses all of the Java programming language class files that make up a package, and converts the package to a standard file format for the binary compatibility of the Java Card platform (CAP file). The Converter also produces an export file.
- Loader: software that transfers the Load File.

During the Phase 7, the TOE must prevent the installation of a package that has not been bytecode verified, or that has been modified after bytecode verification. The loading process requires adding the proof of the origin of the Load File (computed by off-card entity) and verifying it by a Security Domain with Mandated DAP privilege. The following tools are used:

- Compiler: software that generates machine-independent code (bytecode)
- Converter: software that preprocesses all of the Java programming language class files that make up a package, and converts the package to a standard file format for the binary compatibility of the Java Card platform (CAP file). The Converter also produces an export file.
- Verifier: software that performs static checks on the bytecodes of the methods of a CAP file and generates a signature <DAPBlock>.
- Loader: software that transfers the Load File (including the <DAPBlock>).

The bytecode, CAP file, DAP Block can be generated from any software.

2.7.3.1.1 LOADED

The state LOADED is the initial life state just after it has been activated (from the Resident Application) or loaded (from the Resident Application or the Card Manager).

This state is independent of the visibility of the Load File (Get Status command of the Card Manager) and just depends on the presence in the Global Platform registry.

2.7.3.1.2 DELETED

The OPEN may receive a request to delete a Load File. For Load Files in EEPROM, the space previously used to store a physically deleted Load File is reclaimed and may be reused. For Load Files in ROM, a flag definitely prohibits further use. The entry within the Global Platform Registry is also removed, and the OPEN is not required to maintain a record of the deleted Load File's previous existence.

2.7.4 Application Life Cycle

The Application Life Cycle begins when an applet is instantiated in the card. This instantiation may occur directly after loading transaction or alternatively from a Load File which is present on the card. The Application Life Cycle States defined by Global Platform are INSTALLED, SELECTABLE or LOCKED.

Figure 6: (Application Life Cycle), illustrates the state transition diagram for the Application Life Cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

In addition to these states, the Application may define its own Application dependent states. Once the Application reaches the SELECTABLE state, it is responsible for managing the next steps of its own Life Cycle. It may use any Application specific states as long as these do not conflict with the states already defined by Global Platform. The OPEN may not perform these transitions without instruction from the Application and the Application is responsible for defining state transitions and ensuring that these transitioning rules are respected.

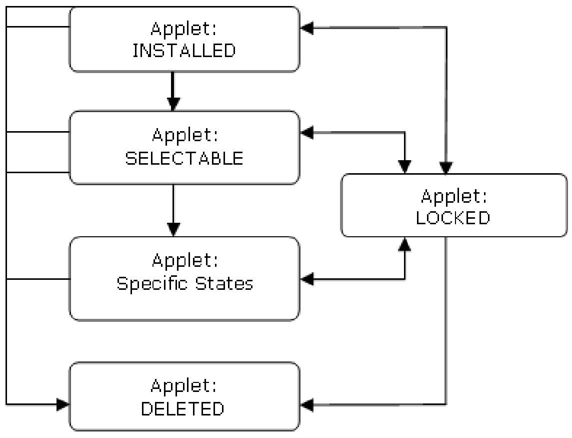


Figure 6: (Application Life Cycle)

2.7.4.1.1 INSTALLED

The INSTALLED state means that the Application executable code has been properly linked and that any necessary memory allocation has taken place. The Application becomes an entry in the Global Platform Registry and this entry is accessible to authenticated off-card entities. The Application is not yet selectable. The installation process is not intended to incorporate personalisation of the Application, which may occur as a separate step.

2.7.4.1.2 SELECTABLE

The SELECTABLE state implies that the applet is able to receive commands from off-card entities. The state transition from INSTALLED to SELECTABLE is irreversible. The Application shall be properly installed and functional before it may be set to the SELECTABLE state. The transition to SELECTABLE may be combined with the Application installation process. The behaviour of the Application in the SELECTABLE state is beyond the scope of this Specification.

2.7.4.1.3 LOCKED

The OPEN or the off-card entity authenticated by the Issuer Security Domain uses the state LOCKED as a security management control to prevent the selection, and therefore the execution, of the Application. If the OPEN detects a threat from within the card and determines that the threat is associated to a particular Application, this Application may be prevented from further selection by the OPEN setting its state to LOCKED. Alternatively, the off-card entity authenticated by the Issuer Security Domain may determine that a particular Application on the card needs to be locked for a business or security reason and may initiate the Application Life Cycle transition via the OPEN. Once the state is LOCKED, only the Issuer Security Domain is allowed to unlock the Application. The OPEN shall ensure that the Application Life Cycle returns to its previous state.

2.7.4.1.4 DELETED

At any point in the Application Life Cycle, the OPEN may receive a request to delete an Application. The space previously used to store a physically deleted Application is reclaimed and may be reused. The entry within the Global Platform Registry is also removed, and the OPEN is not required to maintain a record of the deleted Application's previous existence.

2.7.4.1.5 Application Specific Life Cycle States

These states are Application specific. The behaviour of the Applet during these states is determined by the Applet itself and is beyond the scope of this document. The OPEN does not enforce any control on Application specific Life Cycle State transitions.

3 Common Criteria conformance claim

3.1 Common Criteria

This security Target claims conformance to the Common Criteria version 3.1 revision 3, with the following documents:

1. "Common Criteria for information Technology Security Evaluation, Part 1: Introduction and general model", July 2009, Version 3.1 revision 3 Final
2. "Common Criteria for information Technology Security Evaluation, Part 2: Security Functional requirements", July 2009, Version 3.1 revision 3 Final
3. "Common Criteria for information Technology Security Evaluation, Part 3: Security Assurance requirements", July 2009, Version 3.1 revision 3 Final

The Conformance to the Common Criteria is claims as follows:

Common Criteria	**Conformance rationale**
Part 1	Strict conformance
Part 2	Conformance to the extended part. FCS_RNG.1: “Random number generation”
Part 3	Compliant to EAL5 +, augmented with - ALC_DVS.2: “Sufficiency of security measures” (highest component) - AVA_VAN.5: “Advanced methodical vulnerability analysis” (highest component)

3.2 Protection Profile

This security target claims a demonstrable conformance to:

Java Card Protection Profile – Open Configuration version 3.0 - ANSSI-CC-PP-2010/03_M01

The product is in conformance with the minimum assurance level EAL4+ augmented with ALC_DVS.2 and AVA_VAN.5 described in paragraph 3.2 of the Protection Profile by claiming an evaluation level EAL5+ augmented with ALC_DVS.2 and AVA_VAN.5.

3.3 Conformance claim rationale

This paragraph presents the consistency between the security target and the Java Card System Open configuration profile Protection Profile.

3.3.1 *TOE Type conformance*

The TOE type is in conformance with the TOE type described in the protection profile. For more information on this point, please refer to chapter 2.1 of this security target.

3.3.2 *SPD Statement Consitency*

3.3.2.1 Assets

All assets from the protection profile are included in the security target.

The following assets have been added:

Assets	**Rationale**
D.CONFIG	This asset defines the elements of configuration during the prepresonalization phase
D.SENSITIVE_DATA	This asset describes the set of sensitive data to be protected
D.ARRAY	This asset describes the applets sensitive data
D.JCS_KEYS	This asset describes two cryptographic keys used during the loading of a file in the card
D.BIO	This asset desribes the biometric sensitive data

3.3.2.2 Threats

All threats from the protection profile are included in the security target.

Two additional threats have been added in the security target:

Threats	**Rationale**
T.CONFIGURATION	This threat is directly linked to D.CONFIG
T.CONF_DATA_APPLET	This threat is directly linked to D.ARRAY
T.PATCH_LOADING	This threat is directly linked to patch loading

3.3.2.3 OSPs

All the OSP from the protection profile is included in the security target, no additional OSP have been added.

3.3.2.4 Assumptions

All the assumptions from the protection profile have been added in the security target, except A.DELETION.

A.DELETION has been removed from the security target because the deletion of applets is in the scope of the evaluation, as O.CARD_MANAGEMENT is an objective in this security target.

3.3.3 Objectives

3.3.3.1 Security Objectives for the TOE

All the security objectives for the TOE from the protection profile are included in the security target.

The following security objectives have been added:

Security objectives for the TOE	**Rationale**
O.SCP.SUPPORT	This security objective comes from a security objective for the operational environment
O.SCP.IC	This security objective comes from a security objective for the operational environment
O.SCP.RECOVERY	This security objective comes from a security objective for the operational environment
O.RESIDENT_APPLICATION	This security objective deals with the security of the resident application
O.CARD_MANAGEMENT	This security objective comes from a security objective for the operational environment
O.SECURE_COMPARE	This security objective is linked to D.ARRAY
O.PATCH_LOADING	This security objective is related to patch loading

3.3.3.2 Security Objectives for the Operational Environment

All the security objectives for the operational environment are included in the security target.

Some security objectives for the operational environment has been transformed in security objectives for the TOE, the rationale is presented in the previous chapter.

3.3.4 SFR and SARs Statements consistency

3.3.4.1 SFRs Consistency

All the SFR from the protection profile have been added in the security target.

The following SFR have been added in the security target:

Additional SFR for the Card Manager

SFR	Rationale
FPT_TST.1	Initial startup test in case of future requirement
FCO_NRO.2/CM_DAP	Refinement of the requirements in terms of non-repudiation of the origin to the Card Manager during the DAP process
FIA_AFL.1/CM	Concerns applets composition evaluation
FIA_UAU.1/CM	Concerns smartcard product and composition
FIA_UAU.4/CardIssuer	Prevents from Card Issuer authentication reuse
FIA_UAU.7/CardIssuer	Defines the authentication process
FPR_UNO.1/Key_CM	Prevents from observation of import key operation
FPT_TDC.1/CM	Technical requirement for communication with another trusted IT product
FMT_SMR.2/CM	Defines several roles
FCS_COP.1/CM	Defines the Cryptographic algorithm available to the CM for the Card Issuer authentication

Additional SFR for the resident application

SFR	Rationale
FDP_ACC.2/PP	Access control policy during prepersonalisation
FDP_ACF.1/PP	Access control functions during prepersonalisation
FDP_UCT.1/PP	Precision of the prepersonalisation access control regarding inter-TSF user data confidentiality transfer protection
FDP_ITC.1/PP	Precision of the import of user data during prepersonalisation
FIA_AFL.1/PP	Precision of the authentication failures during prepersonalisation
FIA_UAU.1/PP	Precision of the user accessible functions before user authentication during prepersonalisation
FIA_UID.1/PP	Precision of the user accessible functions before user identification during prepersonalisation
FMT_MSA.1/PP	Precision of the management of the security attributes during prepersonalisation
FMT_SMF.1/PP	Precision of the specification of the management functions during prepersonalisation
FIA_ATD.1/CardManu	Precision of the user attribute during prepersonalisation
FIA_UAU.4/CardManu	Prevents from Card Manufacturer authentication reuse during prepersonalisation
FIA_UAU.7/CardManu	Defines the authentication process of the Card Manufacturer during prepersonalisation
FMT_MOF.1/PP	Management of functions of the TSF during prepersonalisation,

	especially for the resident application
FMT_SMR.2/PP	Restrictions on security roles during prepersonalisation
FMT_MSA.3/PP	Precision of the security attribute intialization during prepersonalisation
FCS_COP.1/PP	Cryptographic operation available during prepersonalisation
FCS_CKM.4/PP	Cryptographic key destruction during prepersonalisation
FDP_UIT.1/PP	Ensures the integrity of the patch loaded
FCS_CKM.1/PP	Provides the MSK diversification
FTP_ITC.1/PP	Defines the trusted channel for the patch and locks loading
FAU_STG.2	Provides the patch identification evidence

Additional SFR for the SmartCard Platform

SFR	Rationale
FPT_PHP.3/SCP	Additional security features are added in the product, using security features of the IC
FPT_FLS.1/SCP	Technical requirement for composition
FPT_RCV.3/SCP	Additional SFR regarding operational objective on the operational environment transformed in security objectives
FPT_RCV.4/SCP	Additional SFR regarding operational objective on the operational environment transformed in security objectives
FRU_FLT.1/SCP	Additional SFR regarding operational objective on the operational environment transformed in security objectives
FCS_RNG.1/SCP	Additional SFR for RNG management
FPR_UNO.1/USE_KEY	Additional SFR for the unobservability of keys
FIA_AFL.1/PIN	Precision of the authentication failures for the PIN
FMT_MTD.2/GP_PIN	Additional SFR for the management of limits on TSF data regarding the GP PIN
FPR_UNO.1/Applet	Additional SFR for the unobservability of array comparison by applets, regarding D.ARRAY
FMT_MTD.1/PIN	Additional SFR for the management of TSF data regarding the PIN
FIA_AFL.1/GP_PIN	Precision of the authentication failures for the GP PIN

Additional SFR for the BIO

SFR	Rationale
FIA_AFL.1/PIN_BIO	Precision of the authentication failures for the PIN BIO
FMT_MTD.1/PIN_BIO	Additional SFR for the management of TSF data regarding the PIN BIO

Additional SFR for the stack control

SFR	Rationale
FDP_ACC.2/RV_Stack	Access control policy for stack control
FDP_ACF.1/RV_Stack	Access control functions for stack control
FMT_MSA.1/RV_Stack	Precision of the Stack access control SFP
FMT_MSA.2/RV_Stack	Precision of the secure security attributes for stack control
FMT_MSA.3/RV_Stack	Precision of the static attribute intialization for stack control
FMT_SMF.1/RV_Stack	Specification of management functions for stack control

Additional SFR for the heap access

SFR	Rationale
FDP_ACC.2/RV_Heap	Access control policy for heap access
FDP_ACF.1/RV_Heap	Access control functions for heap access
FMT_MSA.1/RV_Heap	Precision of the heap access control SFP
FMT_MSA.2/RV_Heap	Precision of the secure security attributes for heap control
FMT_MSA.3/RV_Heap	Precision of the static attribute intialization for heap control
FMT_SMF.1/RV_Heap	Specification of management functions for heap control

Additional SFR for the transient control

SFR	Rationale
FDP_ACC.2/RV_Transient	Access control policy for transient control
FDP_ACF.1/RV_Transient	Access control functions for transient control
FMT_MSA.1/RV_Transient	Precision of the transient access control SFP
FMT_MSA.2/RV_Transient	Precision of the secure security attributes for transient control
FMT_MSA.3/RV_Transient	Precision of the static attribute intialization for transient control
FMT_SMF.1/RV_Transient	Specification of management functions for transient control

4 Security aspects

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

For instance, we will define hereafter the following aspect:

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

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

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

4.1 Confidentiality

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

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

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

4.2 Integrity

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

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

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

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

4.3 Unauthorized executions

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

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

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

#.NATIVE:

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

4.4 Bytecode verification

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

4.4.1 CAP file verification

Bytecode verification includes checking atleast the following properties: (3) bytecodeinstructions represent a legal set of instructions used on the Java Card platform; (4) adequacy of bytecode operands to bytecode semantics; (5) absence of operand stack overflow/underflow; (6) control flow confinement to the current method (that is, no control jumps to outside the method); (7) absence of illegal data conversion and reference forging; (8) enforcement of the private/public access modifiers for class and class members; (9) validity of any kind of reference used in the bytecodes (that is, any pointer to a bytecode, class, method, object, local variable, etc actually points to the beginning of piece of data of the expected kind); (10) enforcement of rules for binary compatibility (full details are given in [R8], [R42], [R43]). The actual set of checks performed by the verifier is implementation-dependent, but shall at least enforce all the “must clauses” imposed in [R8] on the bytecodes and the correctness of the CAP files’ format.

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

Another important aspect to be considered about bytecode verification and application downloading is, first, the assurance that every package required by the loaded applet is indeed on the card, in a binary-compatible version (binary compatibility is explained in [R8] §4.4), second, that the export files used to check and link the loaded applet have the corresponding correct counterpart on the card.

4.4.2 Integrity and authentication

Verification off-card is useless if the application package is modified afterwards. The usage of cryptographic certifications coupled with the verifier in a secure module is a simple means to prevent any attempt of modification between package verification and package installation.

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

Alternatively, the card itself may include a verifier and perform the checks prior to the effective installation of the applet or provide means for the bytecodes to be verified dynamically.

4.4.3 Linking and authentication

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

4.5 Card management

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

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

#.SID:
(1) Users and subjects of the TOE must be identified. (2) The identity of sensitive users and subjects associated with administrative and privileged roles must be particularly protected; this concerns the Java Card RE, the applets registered on the card, and especially the default applet and the currently selected applet (and all other active applets in Java Card System 2.2.x). A change of identity, especially standing for an administrative role (like an applet impersonating the Java Card RE), is a severe violation of the Security Functional Requirements (SFR). Selection controls the access to any data exchange between the TOE and the CAD and therefore, must be protected as well. The loading of a package or any exchange of data through the APDU buffer (which can be accessed by any applet) can lead to disclosure of keys, application code or data, and so on.

#OBJ-DELETION:
(1) Deallocation of objects shouldnot introduce security holes in the form of references pointingto memory zones that are not longer in use, or have been reused for other purposes. Deletion of collection of objects should not bemaliciously used to circumvent the TSFs. (2) Erasure, if deemed successful, shall ensure that the deleted class instance is no longer accessible.

#DELETION:

(1) Deletion of installed applets (or packages) should not introduce security holes in the form of broken references to garbage collected code or data, nor should they alter integrity or confidentiality of remaining applets. The deletion procedure should not be maliciously used to bypass the TSFs. (2) Erasure, if deemed successful, shall ensure that any data owned by the deleted applet is no longer accessible (shared objects shall either prevent deletion or be made inaccessible). A deleted applet cannot be selected or receive APDU commands. Package deletion shall make the code of the package no longer available for execution. (3) Power failure or other failures during the process shall be taken into account in the implementation so as to preserve the SFRs. This does not mandate, however, the process to be atomic. For instance, an interrupted deletion may result in the loss of user data, as long as it does not violate the SFRs.
The deletion procedure and its characteristics (whether deletion is either physical or logical, what happens if the deleted application was the default applet, the order to be observed on the deletion steps) are implementation-dependent. The only commitment is that deletion shall not jeopardize the TOE (or its assets) in case of failure (such as power shortage).
Deletion of a single applet instance and deletion of a whole package are functionally different operations and may obey different security rules. For instance, specific packages can be declared to be undeletable (for instance, the Java Card API packages), or the dependency between installed packages may forbid the deletion (like a package using super classes or super interfaces declared in another package).

4.6 Services

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

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

#.RESOURCES:
The TOE controls the availability ofresources for the applications and enforces quotas and limitations in order to prevent unauthorized denial of service or malfunction of the TSFs. This concerns both execution (dynamic memory allocation) and installation (static memory allocation) of applications and packages.

#.CIPHER:
The TOE shall provide a means to the applications for ciphering sensitive data, for instance, through a programming interface to low-level, highly secure cryptographic services. In particular, those services must support cryptographic algorithms consistent with cryptographic usage policies and standards.

#.KEY-MNGT:
The TOE shall provide a means to securely manage cryptographic keys. This includes: (1) Keys shall be generated in accordance with specified cryptographic key generation algorithms and specified cryptographic key sizes, (2) Keys must be distributed in accordance with specified cryptographic key distribution methods, (3) Keys must be initialized before being used, (4) Keys shall be destroyed in accordance with specified cryptographic key destruction methods.

#.PIN-MNGT:
The TOE shall provide a means to securely manage PIN objects. This includes: (1) Atomic update of PIN value and try counter, (2) No rollback on the PIN-checking function, (3) Keeping the PIN value (once initialized) secret (forinstance, no clear-PIN-reading function), (4) Enhanced protection of PIN’s security attributes (state, try counter…) in confidentiality and integrity.

#.SCP:
The smart card platform must be secure with respect to the SFRs. Then: (1) After a power loss, RF signal loss or sudden card removal prior to completion of some communication protocol, the SCP will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state. (2) It does not allow the SFRs to be bypassed or altered and does not allow access to other low-level functions thanthose made available by the packages of the Java Card API. That includes the protection of its private data and code (against disclosure or modification) from the Java Card System. (3) It provides secure low-level cryptographic processing to the Java Card System. (4) It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism. (5) It allows the Java Card System to store data in “persistent technology memory” or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured andallows for low–level control accesses (segmentation fault detection). (6) It safely transmits low–level exceptions to the TOE (arithmetic exceptions, checksum errors), when applicable. Finally, itis required that (7) the IC is designed in accordance with a well-defined set of policies and standards (for instance, those specified in [R24]), and will be tamper resistant to actually prevent an attacker from extracting or altering security data (like cryptographic keys) by using commonly employed techniques (physical probing and sophisticated analysis of the chip). This especially matters to the management (storage and operation) of cryptographic keys.

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

5 Security Problem Definition

5.1 Assets

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

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

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

5.1.1 User data

D.APP_CODE

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

To be protected from unauthorized modification.

D.APP_C_DATA

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

To be protected from unauthorized disclosure.

D.APP_I_DATA

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

To be protected from unauthorized modification.

D.APP_KEYs

Cryptographic keys owned by the applets.

To be protected from unauthorized disclosure and modification.

D.PIN

Any end-user's PIN.

To be protected from unauthorized disclosure and modification.

5.1.2 TSF data

D.API_DATA

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

To be protected from unauthorized disclosure and modification.

D.CRYPTO

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

To be protected from unauthorized disclosure and modification.

D.JCS_CODE

The code of the Java Card System.

To be protected from unauthorized disclosure and modification.

D.JCS_DATA

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

To be protected from unauthorized disclosure or modification.

D.SEC_DATA

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

To be protected from unauthorized disclosure and modification.

5.1.3 Additional assets

D.CONFIG

The configuration DATA are put at prepersonalisation phase. These elements of configuration have to be loaded securely. To be protected from unauthorized disclosure or modification.

D.SENSITIVE_DATA

The other sensitive data are grouped in the same D.Sensitive Data. The list is presented below:

- D.NB_AUTHENTIC: Number of authentications. This number is specified in the SFR
- D.NB_REMAINTRYOWN: Number of remaining tries for owner PIN. This number is specified in the SFR
- D.NB_REMAINTRYGLB: Number of remaining tries for a global PIN. This number is specified in the SFR
- ASG.CARDREG: Card registry (AS.APID: Applet Identifier (AID), AS.CMID: Card Manager ID (AID))
- ASG.APPRIV: Applet privileges group (Card Manager lock privilege, Card terminate privilege, Default selected privilege, PIN change privilege, Security Domain privilege, Security Domain with DAP verification privilege, Security Domain with Mandated DAP verification privilege)
- AS.AUTH_MSK_STATUS: Authentication MSK Status
- AS.AUTH_CM_STATUS: Authentication CM Status
- AS.CMLIFECYC: Card life cycle state
- AS.MSKKEY: MSK (Manufacturer Secret Key)
- AS.SECURITY_LEVEL: Security levels of a Secure Channel (Confidentiality, Integrity or both) To be protected from unauthorized disclosure and modification.

- D.NB_REMAINTRYOTPINBIO: Number of remaining tries for PIN BIO object. This number is specified by the applet.
- D.TRESHOLDOTPINBIO: Threshold value used for Match On Card comparison. This value is specified by the applet.

D.ARRAY

Applets are enabled to store confidential data. To be protected from unauthorized disclosure and modification.

D.BIO

Any biometric template. To be protected from unauthorized disclosure and modification.

D.JCS_KEYS

AS.KEYSET_VERSION and AS.KEYSET_Value Cryptographic keys used when loading a file into the card. To be protected from unauthorized disclosure and modification.

5.2 Users / Subjects

5.2.1 Additional Users / Subjects

S.RESIDENT_APPLICATION

The resident application

R.personaliser

Card Issuer or card Manufacturer

R.Prepersonaliser

Card manufacturer

U.Card_Issuer

The Card Issuer is the entity that own the card and is ultimately responsible for the behaviour of the card. It is initially the only entity authorized to manage applications through a secure communication channel with the card.

U.Card_Manufacturer

The Card Manufacturer is the entity responsible for producing smart cards on behalf of the Card Issuer.

5.2.2 Miscellaneous

S.ADEL

The applet deletion manager which also acts on behalf of the card issuer. It may be an applet ([R7], §11), but its role asks anyway for a specific treatment from the security viewpoint. This subject is unique and is involved in the ADEL security policy defined in §7.1.3.1.

S.APPLET

Any applet instance.

S.BCV

The bytecode verifier (BCV), which acts on behalf of the verification authority who is in charge of the bytecode verification of the packages. This subject is involved in the PACKAGE LOADING security policy

S.CAD

The CAD represents the actor that requests, by issuing commands to the card, for RMI services. It also plays the role of the off-card entity that communicates with the S.INSTALLER.

S.INSTALLER

The installer is the on-card entity which acts on behalf of the card issuer. This subject is involved in the loading of packages and installation of applets.

S.JCRE

The runtime environment under which Java programs in a smart card are executed.

S.JCVM

The bytecode interpreter that enforces the firewall at runtime.

S.LOCAL

Operand stack of a JCVM frame, or local variable of a JCVM frame containing an object or an array of references.

S.MEMBER

Any object's field, static field or array position.

S.PACKAGE

A package is a namespace within the Java programming language that may contain classes and interfaces, and in the context of Java Card technology, it defines either a user library, or one or several applets.

S.TOE

Source code.

5.3 Threats

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

5.3.1 *CONFIDENTIALITY*

T.CONFID-APPLI-DATA

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

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

T.CONFID-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.CONFID-JCS-DATA

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

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

5.3.2 INTEGRITY

T.INTEG-APPLI-CODE

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-CODE.LOAD

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

Directly threatened asset(s): D.APP_CODE.

T.INTEG-APPLI-DATA

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

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

T.INTEG-APPLI-DATA.LOAD

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

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

T.INTEG-JCS-CODE

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

Directly threatened asset(s): D.JCS_CODE.

T.INTEG-JCS-DATA

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

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

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

5.3.3 IDENTITY USURPATION

T.SID.1

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

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

T.SID.2

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

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

5.3.4 UNAUTHORIZED EXECUTION

T.EXE-CODE.1

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

Directly threatened asset(s): D.APP_CODE.

T.EXE-CODE.2

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

Directly threatened asset(s): D.APP_CODE.

T.EXE-CODE-REMOTE

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

Directly threatened asset(s): D.APP_CODE.

Application Note:

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

T.NATIVE

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

Directly threatened asset(s): D.JCS_DATA.

5.3.5 DENIAL OF SERVICE

T.RESOURCES

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

Directly threatened asset(s): D.JCS_DATA.

5.3.6 CARD MANAGEMENT

T.DELETION

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

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

T.INSTALL

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

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

5.3.7 SERVICES

T.OBJ-DELETION

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

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

5.3.8 MISCELLANEOUS

T.PHYSICAL

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

This threatens all the identified assets.

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

5.3.9 Additional threats

T.CONFIGURATION

The attacker tries to observe or modify configuration information exchanged between the TOE and its environnment. The TOE in this phase must protect itself from modification or theft. Even the field is protected by assurance measures, each operations realised in this phase has to be protected.

T.CONF_DATA_APPLET

The attacker tries to observe the operation of compararison between two byte arrays in order to catch confidential information manipulated.

T.PATCH_LOADING

The attacker tries to avoid the loading of a genuine patch, alter a patch (during loading or once loaded), or to exploit the patch loading mechanism to load unauthenticated code on the TOE, in order to get access to the assets, the TSF data or the TOE user data, or to modify the TSF.

5.4 Organisational Security Policies

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

OSP.VERIFICATION

This policy shall ensure the consistency between the export files used in the verification and those used for installing the verified file. The policy must also ensure that no modification of the file is performed in between its verification and the signing by the verification authority. See #.VERIFICATION for details. OE.VERIFICATION guarantees the correct integrity and authenticity evidences for each application, by means of elements provided by OE.CODE-EVIDENCE.

5.5 Assumptions

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

Due to the Protection Profile and Security Target definition, T.DELETION replaces A.DELETION as O.CARD_MANAGEMENT replaces OE.CARD_MANAGEMENT.

A.APPLET

Applets loaded post-issuance do not contain native methods. The Java Card specification explicitly "does not include support for native methods" ([R8], §3.3) outside the API.

A.VERIFICATION

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

6 Security Objectives

6.1 Security Objectives for the TOE

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

6.1.1 IDENTIFICATION

O.SID

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

6.1.2 EXECUTION

O.FIREWALL

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

O.GLOBAL_ARRAYS_CONFID

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

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

O.GLOBAL_ARRAYS_INTEG

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

O.NATIVE

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

O.OPERATE

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

O.REALLOCATION

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

O.RESOURCES

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

6.1.3 SERVICES

O.ALARM

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

O.CIPHER

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

O.KEY-MNGT

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

O.PIN-MNGT

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

Application Note:

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

O.REMOTE

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

O.TRANSACTION

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

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

6.1.4 *OBJECT DELETION*

O.OBJ-DELETION

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

6.1.5 *APPLET MANAGEMENT*

O.DELETION

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

O.LOAD

The TOE shall ensure that the loading of a package into the card is safe. Besides, for code loaded post-issuance, the TOE shall verify the integrity and authenticity evidences generated during the verification of the application package by the verification authority. This verification by the TOE shall occur during the loading or later during the install process.

Application Note:

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

O.INSTALL

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

6.1.6 Additional security objectives for the TOE

Four security objectives for the operational environment defined in the PP JCS have been transformed in security objectives for the TOE:

- OE.SCP.IC
- OE.SCP.SUPPORT
- OE.SCP.RECOVERY
- OE.CARD_MANAGEMENT

O.SCP.SUPPORT

The TOE shall support the following functionalities:

- It does not allow the TSFs to be bypassed or altered and does not allow access to other low-level functions than those made available by the packages of the API. That includes the protection of its private data and code (against disclosure or modification) from the Java Card System.
- It provides secure low-level cryptographic processing to the Java Card System and Global Platform.
- It supports the needs for any update to a single persistent object or class field to be atomic, and possibly a low-level transaction mechanism.
- It allows the Java Card System to store data in "persistent technology memory" or in volatile memory, depending on its needs (for instance, transient objects must not be stored in non-volatile memory). The memory model is structured and allows for low-level control accesses (segmentation fault detection).

O.SCP.IC

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

O.SCP.RECOVERY

If there is a loss of power, or if the smart card is withdrawn from the CAD while an operation is in progress, the SCP must allow the TOE to eventually complete the interrupted operation successfully, or recover to a consistent and secure state. The smart card platform must be secure with respect to the SFRs. Then after a power loss or sudden card removal prior to completion of some communiication protocol, the SCP will allow the TOE on the next power up to either complete the interrupted operation or revert to a secure state.

O.RESIDENT_APPLICATION

This objective concerns the resident application. It provides a native code application, with a basic main dispatcher to receive card commands and dispatch them to the application and module functions that implement the application commands. It also deals with the Card Manufacturer authentication and logical channels management. The dispatcher is always activated. Some card commands (for administration) are only available during prepersonalisation phase. It ensures the personaliser authentication before allowing operations in writing of the resident application.

O.CARD_MANAGEMENT

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

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

O.SECURE_COMPARE

The TOE shall provide to applet a means to securely compare two byte arrays.

O.PATCH_LOADING

The TOE shall provide a secure patch code loading mechanism.

6.2 Security objectives for the Operational Environment

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

Four security objectives for the operational environment from the PP JCS have been transformed in security objectives for the TOE:

- OE.SCP.SUPPORT
- OE.SCP.IC
- OE.SCP.RECOVERY
- OE.CARD_MANAGEMENT

OE.APPLET

No applet loaded post-issuance shall contain native methods.

OE.VERIFICATION

All the bytecodes shall be verified at least once, before the loading, before the installation or before the execution, depending on the card capabilities, in order to ensure that each bytecode is valid at execution time. See #.VERIFICATION for details. Additionally, the applet shall follow all the recommendations, if any, mandated in the platform guidance for maintaining the isolation property of the platform. The list of applications embedded in the product is defined in the following document: [R32]

Application Note:

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

OE.CODE-EVIDENCE

For application code loaded pre-issuance, evaluated technical measures implemented by the TOE or audited organizational measures must ensure that loaded application has not been changed since the code verifications required in OE.VERIFICATION. For application code loaded post-issuance and verified off-card according to the requirements of OE.VERIFICATION, the verification authority shall provide digital evidence to the TOE that the application code has not been modified after the code verification and that he is the actor who performed code verification. For application code loaded post-issuance and partially or entirely verified on-card, technical measures must ensure that the verification required in OE.VERIFICATION are performed. On-card bytecode verifier is out of the scope of this Security Target.

7 Extended Requirements

7.1 Extended Families

7.1.1 *Extended Family FCS_RNG - FCS_RNG: Random Number Generation*

7.1.1.1 Description

This family defines quality requirements for the generation of random numbers which are intended to be used for cryptographic purposes.

7.1.1.2 Extended Components

7.1.1.2.1 Extended Component FCS_RNG.1

Description

A physical random number generator (RNG) produces the random number by a noise source based on physical random processes. A non-physical true RNG uses a noise source based on non-physical random processes like human interaction (key strokes, mouse movement). A deterministic RNG uses a random seed to produce a pseudorandom output. A hybrid RNG combines the principles of physical and deterministic RNGs.

Family behaviour:

This family defines quality requirements for the generation of random numbers which are intended to be use for cryptographic purposes.

Component levelling:

Generation of random numbers requires that random numbers meet a defined quality metric.

Management:

There are no management activities foreseen

Audit:

There are no actions defined to be auditable

Hierarchical to:

No other components

Definition

FCS_RNG.1 Random Number Generation

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

FCS_RNG.1.2 The TSF shall provide random numbers that meet [assignment: a defined quality metric].

Dependencies: No dependencies.

8 Security Requirements

8.1 Security Functional Requirements

This section states the security functional requirements for the Java Card System - Open configuration. For readability and for compatibility with the original Java Card System Protection Profile Collection - Standard 2.2 Configuration [R45], requirements are arranged into groups. All the groups defined in the table below apply to this Security Target.

Group	Description
Core with Logical Channels (*CoreG_LC*)	The CoreG_LC contains the requirements concerning the runtime environment of the Java Card System implementing logical channels. This includes the firewall policy and the requirements related to the Java Card API. Logical channels are a Java Card specification version 2.2 feature. This group is the union of requirements from the Core (*CoreG*) and the Logical channels (*LCG*) groups defined in [R44] (cf. Java Card System Protection Profile Collection [R44]).
Installation (*InstG*)	The InstG contains the security requirements concerning the installation of post-issuance applications. It does not address card management issues in the broad sense, but only those security aspects of the installation procedure that are related to applet execution.
Applet deletion (*ADELG*)	The ADELG contains the security requirements for erasing installed applets from the card, a feature introduced in Java Card specification version 2.2.
Remote Method Invocation (RMI)	The RMIG contains the security requirements for the remote method invocation feature, which provides a new protocol of communication between the terminal and the applets. This was introduced in Java Card specification version 2.2.
Object deletion (*ODELG*)	The ODELG contains the security requirements for the object deletion capability. This provides a safe memory recovering mechanism. This is a Java Card specification version 2.2 feature.
Secure carrier (*CarG*)	The CarG group contains minimal requirements for secure downloading of applications on the card. This group contains the security requirements for preventing, in those configurations that do not support on-card static or dynamic bytecode verification, the installation of a package that has not been bytecode verified, or that has been modified after bytecode verification.

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

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

Object	Description

O.APPLET	Any installed applet, its code and data
O.CODE_PKG	The code of a package, including all linking information. On the Java Card platform, a package is the installation unit
O.JAVAOBJECT	Java class instance or array. It should be noticed that KEYS, PIN, arrays and applet instances are specific objects in the Java programming language
O.REMOTE_MTHD	A method of a remote interface
O.REMOTE_OBJ	A remote object is an instance of a class that implements one (or more) remote interfaces. A remote interface is one that extends, directly or indirectly, the interface java.rmi.Remote ([R6])
O.RMI_SERVICE	These are instances of the class javacardx.rmi.RMIService. They are the objects that actually process the RMI services.
O.ROR	A remote object reference. It provides information concerning: (i) the identification of a remote object and (ii) the Implementation class of the object or the interfaces implemented by the class of the object. This is the object's information to which the CAD can access

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

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

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

Security attribute	Description/Value
Active Applets	The set of the active applets' AIDs. An active applet is an applet that is selected on at least one of the logical channels.
Applet Selection Status	"Selected" or "Deselected".
Applet's version number	The version number of an applet (package) indicated in the export file.
Class	Identifies the implementation class of the remote object.
Context	Package AID or "Java Card RE".
COD Context attribute	Delimits the space occupied in volatile memory by the data of the CLEAR_ON_DESELECT transient arrays of a package
COR Context attribute	Delimits the space occupied in volatile memory by the data of the CLEAR_ON_RESET transient arrays of a package
Current Frame Context	The lower and upper Boundary of the local variables area on the stack frame for a method and the lower and upper Boundary of the operand stack area on the stack frame for a method
Currently Active Context	Package AID or "Java Card RE".
Dependent package AID	Allows the retrieval of the Package AID and Applet's version number ([R8], §4.5.2).
ExportedInfo	Boolean (indicates whether the remote object is exportable or not).
Identifier	The Identifier of a remote object or method is a number that uniquely identifies the remote object or method, respectively.
LC Selection Status	Multiselectable, Non-multiselectable or "None".
LifeTime	CLEAR_ON_DESELECT or PERSISTENT or CLEAR_ON_RESET
Object Boundary	Delimits the space occupied by an object in the heap
Owner	The Owner of an object is either the applet instance that created the object or the package (library) where it has been defined (these latter objects can only be arrays that initialize static fields of the package). The owner of a remote object is the applet instance that created the object.
Package AID	The AID of each package indicated in the export file.
Package Boundary	Delimits the space occupied by the code and the static fields of a package

Security attribute	Description/Value
Program Counter	Position of the next Bytecode to execute
Registered Applets	The set of AID of the applet instances registered on the card.
Remote	An object is Remote if it is an instance of a class that directly or indirectly implements the interface java.rmi.Remote.
Resident Packages	The set of AIDs of the packages already loaded on the card.
Returned References	The set of remote object references that have been sent to the CAD during the applet selection session. This attribute is implementation dependent.
Selected Applet Context	Package AID or "None".
Sharing	Standards, SIO, Java Card RE entry point or global array.
Stack Pointer	Position of the next free slot in the stack
Static Fields	Static fields of a package
Static References	Static fields of a package may contain references to objects. The Static References attribute records those references.

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

Operation	Description
OP.ARRAY_ACCESS (O.JAVAOBJECT, field)	Read/Write an array component.
OP.CREATE (Sharing, LifeTime)	Creation of an object (new or makeTransient call).
OP.DELETE_APPLET (O.APPLET,...)	Delete an installed applet and its objects, either logically or physically.
OP.DELETE_PCKG (O.CODE_PKG,...)	Delete a package, either logically or physically.
OP.DELETE_PCKG_APPLET (O.CODE_PKG,...)	Delete a package and its installed applets, either logically or physically.
OP.FLOW (O.CODE_PKG)	Any operation that modify the execution flow
OP.GET_ROR (O.APPLET,...)	Retrieves the initial remote object reference of a RMI based applet. This reference is the seed which the CAD client application needs to begin remote method invocations.
OP.IMPORT_KEY	Import of the keys

Operation	Description
OP.INSTANCE_FIELD (O.JAVAOBJECT, field)	Read/Write a field of an instance of a class in the Java programming language.
OP.INVK_INTERFACE (O.JAVAOBJECT, method, arg1,...)	Invoke an interface method.
OP.INVK_VIRTUAL (O.JAVAOBJECT, method, arg1,...)	Invoke a virtual method (either on a class instance or an array object).
OP.INVOKE (O.RMI_SERVICE,...)	Requests a remote method invocation on the remote object.
OP.JAVA (...)	Any access in the sense of [R7], §6.2.8. It stands for one of the operations OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW, OP.TYPE_ACCESS.
OP.LOCAL_STACK_ACCESS (...)	Any operation that read or write the local stack
OP.OPERAND_STACK_ACCESS (...)	Any operation that push or pop items on the operand stack
OP.PUT (S1,S2,I)	Transfer a piece of information I from S1 to S2.
OP.RET_RORD (S.JCRE,S.CAD,I.RORD)	Send a remote object reference descriptor to the CAD.
OP.STATIC_FIELD (O.CODE_PKG, field)	Read/Write a static field of a class in the JAVA prgramming language
OP.THROW (O.JAVAOBJECT)	Throwing of an object (athrow, see [R7], §6.2.8.7).
OP.TYPE_ACCESS (O.JAVAOBJECT, class)	Invoke checkcast or instanceof on an object in order to access to classes (standard or shareable interfaces objects).
Cardholder Authentication	Authentication of the cardholder
U.Card_Issuer authentication	Authentication of U.Card_Issuer

8.1.1 CoreG_LC Security Functional Requirements

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

8.1.1.1 Firewall Policy

FDP_ACC.2/FIREWALL Complete access control

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

Refinement:

The operations involved in the policy are:

- OP.CREATE,
- OP.INVK_INTERFACE,

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

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

Application Note:

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

FDP_ACF.1/FIREWALL Security attribute based access control

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

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

.

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

- **R.JAVA.1 ([R7], §6.2.8): S.PACKAGE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE, OP.THROW or OP.TYPE_ACCESS upon any O.JAVAOBJECT whose Sharing attribute has value "JCRE entry point" or "global array".**
- **R.JAVA.2 ([R7], §6.2.8): S.PACKAGE may freely perform OP.ARRAY_ACCESS, OP.INSTANCE_FIELD, OP.INVK_VIRTUAL, OP.INVK_INTERFACE or OP.THROW upon any O.JAVAOBJECT whose Sharing attribute has value "Standard" and whose Lifetime attribute has value "PERSISTENT" only if O.JAVAOBJECT's Context attribute has the same value as the active context.**
- **R.JAVA.3 ([R7], §6.2.8.10): S.PACKAGE may perform OP.TYPE_ACCESS upon an O.JAVAOBJECT whose Sharing attribute has value "SIO" only if O.JAVAOBJECT is being cast into (checkcast) or is being verified as being an instance of (instanceof) an interface that extends the Shareable interface.**
- **R.JAVA.4 ([R7], §6.2.8.6): S.PACKAGE may perform OP.INVK_INTERFACE upon an O.JAVAOBJECT whose Sharing attribute has the value "SIO", and whose Context attribute has the value "Package AID", only if the invoked interface method extends the Shareable interface and one of the following conditions applies:**
  - **a) The value of the attribute Selection Status of the package whose AID is "Package AID" is "Multiselectable",**

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

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

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

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

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

FDP_IFC.1/JCVM Subset information flow control

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

FDP_IFF.1/JCVM Simple security attributes

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

Subjects	Security attributes
S.JCVM	Currently Active Context

.

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

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

FDP_IFF.1.3/JCVM The TSF shall enforce the **none**.

FDP_IFF.1.4/JCVM The TSF shall explicitly authorise an information flow based on the following rules: **none**.

FDP_IFF.1.5/JCVM The TSF shall explicitly deny an information flow based on the following rules: **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 SFP** to restrict the ability to **modify** the security attributes **Selected Applet Context** to **the Java Card RE**.

FMT_MSA.1/JCVM Management of security attributes

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

FMT_MSA.2/FIREWALL_JCVM Secure security attributes

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

FMT_MSA.3/FIREWALL Static attribute initialisation

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

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

FMT_MSA.3/JCVM Static attribute initialisation

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

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

FMT_SMF.1 Specification of Management Functions

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

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

FMT_SMR.1 Security roles

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

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

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

8.1.1.2 Application Programming Interface

The following SFRs are related to the Java Card API.

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

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

FCS_CKM.1 Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **see table below** and specified cryptographic key sizes **see table below** that meet the following: **see table below:**

Cryptographic key generation algorithm	Cryptographic key size	List of standards
TDES	112 bits or 168 bits	FIPS PUB 46-3 (ANSI X3.92),

		FIPS PUB 81
ECKeyP	from 160 to 521 bits	IEEE Std 1363a-2004 [R27]
RSA	from 64 to 2048 bits with a step of 32-bit	ANSI X9.31
AES	from 128 to 256 bits with a step of 64 bits	FIPS PUB 197
GP Keys - TDES (ECB)	112 bits	GP2.1
GP Keys – AES (ECB)	128, 192, 256 bits	GP2.1
GP Keys – AES (ECB)	128 bits	Proprietary SCPF3

.

FCS_CKM.2 Cryptographic key distribution

FCS_CKM.2.1 The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method **setKey** that meets the following: **Java Card API [R6] specification and setEncKey/setMacKey in the class ISOSecureMessaging (Package "com.oberthurcs.javacard.utilSM")**.

FCS_CKM.3 Cryptographic key access

FCS_CKM.3.1 The TSF shall perform **the following types of cryptographic key access** in accordance with a specified cryptographic key access method **see refinement below** that meets the following:

- o **Packages "javacard.security" and "javacard.crypto"**
- o **Package "com.oberthurcs.javacard.utilSM"**
- o **Package "org.Global Platform"**
- o **"Java Card JCRE" specification [JCRE]**
- o **"Global Platform Card 2.2" specification [R12]**
- o **"Java Card API" specification [R6]**.

Refinement:

Type of cryptographic key access Cryptographic key access methods (or commands)

DES
The following commands: PUT_KEY, EXTERNAL AUTHENTICATE, INITIALIZE UPDATE. The following SecureChannel key access methods Unwrap, wrap, decryptData, encryptData, resetSecurity The following ISOSecureMessaging key access methods reset, setEncKey, setKeyFormat, setMacKey, unwrap_LDS, wrap_LDS, wrapLong, wrapLongFinal, wrapLongInit, wrapSW_LDS, setKeyFormat The following "APICrypto" key access methods: Key.clearKey, DES.getKey, DES.setKey, Signature.init, Signature.update, Signature.sign, Signature.verify, Cipher.init, Cipher.update, Cipher.doFinal

AES
The following commands: PUT_KEY, EXTERNAL AUTHENTICATE, INITIALIZE UPDATE, The following "ProviderSecurityDomain" key access methods: decryptVerifyKey, openSecureChannel, unwrap,

verifyExternalAuthenticate The following SecureChannel key access methods Unwrap, wrap, decryptData, encryptData, resetSecurity The following ISOSecureMessaging key access methods reset, setEncKey, setKeyFormat, setMacKey, unwrap_LDS, wrap_LDS, wrapLong, wrapLongFinal, wrapLongInit, wrapSW_LDS, setKeyFormat The following "APICrypto" key access methods: Key.clearKey, AES.getKey, AES.setKey, Signature.init, Signature.update, Signature.sign, Signature.verify, Cipher.init, Cipher.update, Cipher.doFinal

RSA
The following commands: PUT_KEY, LOAD The following "ProviderSecurityDomain" key access methods: DecryptVerifyKey, The following "APICrypto" key access methods: Key.clearKey, RSAPrivateCRTKey.setP, RSAPrivateCRTKey.setQ, RSAPrivateCRTKey.setPQ, RSAPrivateCRTKey.setDP1, RSAPrivateCRTKey.setDQ1, RSAPrivateCRTKey.getP, RSAPrivateCRTKey.getQ, RSAPrivateCRTKey.getPQ, RSAPrivateCRTKey.getDP1, RSAPrivateCRTKey.getDQ1, RSAPrivateKey.setModulus, RSAPrivateKey.setExponent, RSAPrivateKey.getModulus, RSAPrivateKey.getExponent, RSAPublicKey.setModulus, RSAPublicKey.setExponent, RSAPublicKey.getModulus, RSAPublicKey.getExponent, Signature.init, Signature.update, Signature.sign, Signature.verify, Cipher.init, Cipher.update, Cipher.doFinal

ECkeyP
The following "APICrypto" key access methods: Key.clearKey, ECPrivateKey.setFieldFP, ECPrivateKey.setA, ECPrivateKey.setB, ECPrivateKey.setG, ECPrivateKey.setR, ECPrivateKey.setK, ECPrivateKey.getField, ECPrivateKey.getA, ECPrivateKey.getB, ECPrivateKey.getG, ECPrivateKey.getR, ECPrivateKey.getK, ECPrivateKey.setS, ECPrivateKey.getS, ECPublicKey.setFieldFP, ECPublicKey.setA, ECPublicKey.setB, ECPublicKey.setG, ECPublicKey.setR,ECPublicKey.setK, ECPublicKey.getField, ECPublicKey.getA, ECPublicKey.getB, ECPublicKey.getG, ECPublicKey.getR, ECPublicKey.getK, ECPublicKey.setW, ECPublicKey.getW, Signature.init, Signature.update, Signature.sign, Signature.verify KeyAgreement.init, KeyAgreement.generateSecret

Application Note:

- The keys can be accessed as specified in [R6] Key class.
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([R6]).

FCS_CKM.4 Cryptographic key destruction

FCS_CKM.4.1 The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method **The keys are reset in accordance with [R6] in class Key with the method clearKey(). Any access to a cleared key attempting to use it for ciphering or signing shall throw an exception** that meets the following: **"Java Card API" specification [R6]. The keys used in class ISOSecureMessaging (Package "com.oberthurcs.javacard.utilSM") are classes Key that meets the following: "Java Card API" specification [R6]. The methods 'reset' and 'setKeyFormat' call the method key.clearKey() for clearing the value of each key**.

Application Note:

- The keys are reset as specified in [R6] Key class, with the method clearKey(). Any access to a cleared key for ciphering or signing shall throw an exception.
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([R6]).

FCS_COP.1 Cryptographic operation

FCS_COP.1.1 The TSF shall perform **see table** in accordance with a specified cryptographic algorithm **see table** and cryptographic key sizes **see table** that meet the following: **see below:**

Cryptographic operation	Cryptographic algorithm	Key size	List of standards
signature, signature's verification, encryption and decryption	DES - TDES	56, 112 or 168 bits	FIPS PUB 46-3, ANSI X3.92, FIPS PUB 81, ISO/IEC 9797(1999), Data integrity mechanism [R17]
signature, signature's verification, encryption and decryption	AES	from 128 to 256 bits with a step of 64 bits	FIPS PUB 197 SP800-38B (CMAC)
signature, signature's verification, encryption and decryption	RSA CRT, RSA SFM	from 64 to 2048 bits with a step of 32-bit	ANSI X9.31, ISO/IEC 9796-1, annex A, section A.4 and A.5, and annex C, PKCS#1
Hash functions	SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	no keys	Secure Hash Standard, FIPS PUB 180-3
signature, signature's verification, encryption and decryption	ECDSA	160 to 521 bits	ANSI X9.62-1998
Key agreement	ECDH	160 to 521 bits	BSI TR 03111 v1.11 IEEE P1363
Checksum	CRC	16 and 32 bits	ISO3309_CRC16 ISO3309_CRC32

Refinement:

TDES (IC)/OT has developed the algorithm using HW DES module/TDES encryption and decryption/Triple Data Encryption (TDES)/56/112/168-bits/E-D-E triple- encryption implementation of the Data Encryption Standard, FIPS PUB 46-3, 25 Oct. 1999

SHA /OT has developed the algorithm/Hash function/SHA-1/No cryptographic key/Secure Hash Standard, Federal Information Processing Standards Publication 180-3, 2008, october

SHA /OT has developed the algorithm/Hash function/SHA-224/No cryptographic key/Secure Hash Standard, Federal Information Processing Standards Publication 180-3, 2008, october

SHA /OT has developed the algorithm/Hash function/SHA-256/No cryptographic key/Secure Hash Standard, Federal Information Processing Standards Publication 180-3, 2008, october

SHA /OT has developed the algorithm/Hash function/SHA-384/No cryptographic key/Secure Hash Standard, Federal Information Processing Standards Publication 180-3, 2008, october

SHA /OT has developed the algorithm/Hash function/SHA-512/No cryptographic key/Secure Hash Standard, Federal Information Processing Standards Publication 180-3, 2008, october

KG /OT has developed the algorithm using HW PK accelerator/Key Generator//Between 1024 bits to 2048 bits/

RSA without CRT /OT has developed the algorithm using HW PK accelerator/Data Encryption and Decryption/RSA Without CRT Data /Between 1024 bits to 2048 bits/PKCS#1 V2.0; 1st October, 1998

RSA with CRT /OT has developed the algorithm using HW PK accelerator/Data Encryption and Decryption/RSA With CRT Data /Between 1024 bits and 2048 bits/PKCS#1 V2.0; 1st October, 1998
RNG/OT has developed the algorithm using HW RNG as seed/Random generator//No cryptographic key/FIPS SP800-90, 2007, March

AES/OT has developed the algorithm/Data encryption / decryption//128/192/256 bits/FIPS PUB 197, 2001, November

Application Note:

- The TOE shall provide a subset of cryptographic operations defined in [R6] (see javacardx.crypto.Cipher and javacardx.security packages).
- This component shall be instantiated according to the version of the Java Card API applicable to the security target and the implemented algorithms ([R6]).

FDP_RIP.1/ABORT Subset residual information protection

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

Application Note:

The events that provoke the de-allocation of a transient object are described in [R7], §5.1.

FDP_RIP.1/APDU Subset residual information protection

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

Application Note:

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

FDP_RIP.1/bArray Subset residual information protection

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

Application Note:

A resource is allocated to the bArray object when a call to an applet's install() method is performed. There is no conflict with FDP_ROL.1 here because of the bounds on the rollback mechanism

(FDP_ROL.1.2/FIREWALL): the scope of the rollback does not extend outside the execution of the install() method, and the de-allocation occurs precisely right after the return of it.

FDP_RIP.1/KEYS Subset residual information protection

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

Application Note:

- The javacard.security & javacardx.crypto packages do provide secure interfaces to the cryptographic buffer in a transparent way. See javacard.security.KeyBuilder and Key interface of [R6].

FDP_RIP.1/TRANSIENT Subset residual information protection

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

Application Note:

- The events that provoke the de-allocation of any transient object are described in [R7], §5.1.
- The clearing of CLEAR_ON_DESELECT objects is not necessarily performed when the owner of the objects is deselected. In the presence of multiselectable applet instances, CLEAR_ON_DESELECT memory segments may be attached to applets that are active in different logical channels. Multiselectable applet instances within a same package must share the transient memory segment if they are concurrently active ([R7], §4.2.

FDP_ROL.1/FIREWALL Basic rollback

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

FDP_ROL.1.2/FIREWALL The TSF shall permit operations to be rolled back within the **scope of a select(), deselect(), process(), install() or uninstall() call, notwithstanding the restrictions given in [R7], §7.7, within the bounds of the Commit Capacity ([R7], §7.8), and those described in [R6]**.

Application Note:

Transactions are a service offered by the APIs to applets. It is also used by some APIs to guarantee the atomicity of some operation. This mechanism is either implemented in Java Card platform or relies on the transaction mechanism offered by the underlying platform. Some operations of the API are not conditionally updated, as documented in [R6] (see for instance, PIN-blocking, PIN-checking, update of Transient objects).

8.1.1.3 Card Security Management

FAU_ARP.1 Security alarms

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

- o **throw an exception,**
- o **lock the card session,**
- o **reinitialize the Java Card System and its data**

upon detection of a potential security violation.

Refinement:

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

- CAP file inconsistency,
- typing error in the operands of a bytecode,
- applet life cycle inconsistency,
- card tearing (unexpected removal of the Card out of the CAD) and power failure,
- abort of a transaction in an unexpected context,
- violation of the Firewall or JCVM SFPs,
- unavailability of resources,
- array overflow

FDP_SDI.2 Stored data integrity monitoring and action

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

FDP_SDI.2.2 Upon detection of a data integrity error, the TSF shall **increase counter of the Killcard file. If the maximum is reached the killcard is launched**.

Application Note:

The following data persistently stored by TOE have the user data attribute "integrityControlledData ":

- PINs (i.e. objects instance of class OwnerPin or subclass of interface PIN)
- Keys (i.e. objects instance of classes implemented the interface Key)
- SecureStores (i.e. objects instance of class SecureStore)
- Packages Java Card
- Patchs
- BIOMETRIC_DATA

FPR_UNO.1 Unobservability

FPR_UNO.1.1 The TSF shall ensure that **any user** are unable to observe the operation **Cardholder authentication** on **D.PIN** by **no user and no subject**.

Application Note:

Although it is not required in [R7] specifications, the non-observability of operations on sensitive information such as keys appears as impossible to circumvent in the smart card world. The precise list of operations and objects is left unspecified, but should at least concern secret keys and PIN codes when they exists on the card, as well as the cryptographic operations and comparisons performed on them.

FPT_FLS.1 Failure with preservation of secure state

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

Application Note:

The Java Card RE Context is the Current context when the Java Card VM begins running after a card reset ([R7], §6.2.3) or after a proximity card (PICC) activation sequence ([R7]). Behaviour of the TOE on power loss and reset is described in [R7], §3.6 and §7.1. Behaviour of the TOE on RF signal loss is described in [R7], §3.6.1.

FPT_TDC.1 Inter-TSF basic TSF data consistency

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

FPT_TDC.1.2 The TSF shall use

- **the rules defined in [R8] specification,**
- **the API tokens defined in the export files of reference implementation,**

when interpreting the TSF data from another trusted IT product.

Application Note:

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

8.1.1.4 AID Management

FIA_ATD.1/AID User attribute definition

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

- **Package AID,**
- **Applet's version number,**
- **Registered applet AID,**
- **Applet Selection Status ([R8], §6.5)**.

Refinement:

"Individual users" stand for applets.

FIA_UID.2/AID User identification before any action

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

Application Note:

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

FIA_USB.1/AID User-subject binding

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

FIA_USB.1.2/AID The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: **rules are defined in FDP_ACC.2/Firewall and FDP_ACF.1/Firewall**.

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

Application Note:

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

FMT_MTD.1/JCRE Management of TSF data

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

Application Note:

- The installer and the Java Card RE manage other TSF data such as the applet life cycle or CAP files, but this management is implementation specific. Objects in the Java programming language may also try to query AIDs of installed applets through the lookupAID(...) API method.
- The installer, applet deletion manager or even the card manager may be granted the right to modify the list of registered applets' AIDs in specific implementations (possibly needed for installation and deletion; see #.DELETION and #.INSTALL).

FMT_MTD.3/JCRE Secure TSF data

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

8.1.2 InstG Security Functional Requirements

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

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

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

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

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

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

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

Package loading is allowed only if, for each dependent package, its AID attribute is equal to a resident package AID attribute, the major (minor) Version attribute associated to the

dependent package is lesser than or equal to the major (minor) Version attribute associated to the resident package ([R8], §4.5.2)..

Application Note:

FDP_ITC.2.1/Installer:

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

FDP_ITC.2.3/Installer:

- The format of the CAP file is precisely defined in [R8] specifications; it contains the user data (like applet's code and data) and the security attributes altogether. Therefore there is no association to be carried out elsewhere.

FDP_ITC.2.4/Installer:

- Each package contains a package Version attribute, which is a pair of major and minor version numbers ([R8], §4.5). With the AID, it describes the package defined in the CAP file. When an export file is used during preparation of a CAP file, the versions numbers and AIDs indicated in the export file are recorded in the CAP files ([R8], §4.5.2): the dependent packages Versions and AIDs attributes allow the retrieval of these identifications. Implementation-dependent checks may occur on a case-by-case basis to indicate that package files are binary compatible. However, package files do have "package Version Numbers" ([R8]) used to indicate binary compatibility or incompatibility between successive implementations of a package, which obviously directly concern this requirement.

FDP_ITC.2.5/Installer:

- A package may depend on (import or use data from) other packages already installed. This dependency is explicitly stated in the loaded package in the form of a list of package AIDs.
- The intent of this rule is to ensure the binary compatibility of the package with those already on the card ([R8], §4.4).
- The installation (the invocation of an applet's install method by the installer) is implementation dependent ([R7], §11.2).
- Other rules governing the installation of an applet, that is, its registration to make it SELECTable by giving it a unique AID, are also implementation dependent (see, for example, [R7], §11).

FMT_SMR.1/Installer Security roles

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

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

FPT_FLS.1/Installer Failure with preservation of secure state

FPT_FLS.1.1/Installer The TSF shall preserve a secure state when the following types of failures occur: **the installer fails to load/install a package/applet as described in [R7] §11.1.4**.

Application Note:

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

FPT_RCV.3/Installer Automated recovery without undue loss

FPT_RCV.3.1/Installer When automated recovery from **An applet (i.e. a package) is considered as loaded, once its reference is written in the list of the loaded packages (i.e. instantiated applets). This is the ultimate stage of the applet/package installation, done when everything has succeeded before (verification, initialization, object instantiation). If an error occurs before registration, everything must be rolled back. For package installation, the garbage collector will automatically remove the package code since we stopped installation before the package recording. For applet installation, we mainly relies on garbage collector, as it is done for package, to remove the applet instance and AID objects (since the applet is not on the root of persistence, these objects are unreachable). On applet installation, its install method is called which can lead to change the states of the VM objects. To rollback the modifications eventually made in field of other persistent objects, the installation is surrounded by a transaction (that is aborted). Finally, we have additional mechanisms to rollback modifications eventually done in the field of transient arrays since they are not covered but the transaction (volatile data is not in the scope of Java Card transaction)** is not possible, the TSF shall enter a maintenance mode where the ability to return to a secure state is provided.

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

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

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

Application Note:

FPT_RCV.3.1/Installer:

- This element is not within the scope of the Java Card specification, which only mandates the behaviour of the Java Card System in good working order. Further details on the "maintenance mode" shall be provided in specific implementations. The following is an excerpt from [R2], p298: In this maintenance mode normal operation might be impossible or severely restricted, as otherwise insecure situations might occur. Typically, only authorised users should be allowed access to this mode but the real details of who can access this mode is a function of FMT: Security management. If FMT: Security management does not put any controls on who can access this mode, then it may be acceptable to allow any user to restore the system if the TOE enters such a state. However, in practice, this is probably not desirable as the user restoring the system has an opportunity to configure the TOE in such a way as to violate the SFRs.

FPT_RCV.3.2/Installer:

- Should the installer fail during loading/installation of a package/applet, it has to revert to a "consistent and secure state". The Java Card RE has some clean up duties as well; see [R7], §11.1.5 for possible scenarios. Precise behaviour is left to implementers. This component shall include among the listed failures the deletion of a package/applet. See ([R7], 11.3.4) for possible scenarios. Precise behaviour is left to implementers.
- Other events such as the unexpected tearing of the card, power loss, and so on, are partially handled by the underlying hardware platform (see [R24]) and, from the TOE's side, by events "that clear transient objects" and transactional features. See FPT_FLS.1.1, FDP_RIP.1/TRANSIENT, FDP_RIP.1/ABORT and FDP_ROL.1/FIREWALL.

FPT_RCV.3.3/Installer:

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

8.1.3 ADELG Security Functional Requirements

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

FDP_ACC.2/ADEL Complete access control

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

Refinement:

The operations involved in the policy are:

- OP.DELETE_APPLET,
- OP.DELETE_PCKG,
- OP.DELETE_PCKG_APPLET.

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

FDP_ACF.1/ADEL Security attribute based access control

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

Subject/Object	Attributes
S.JCVM	Active Applets
S.JCRE	Selected Applet Context, Registered Applets, Resident Packages
O.CODE_PKG	Package AID, Dependent Package AID, Static References
O.APPLET	Applet Selection Status
O.JAVAOBJECT	Owner, Remote

.

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

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

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

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

- **R.JAVA.14 ([R7], §11.3.4.1, Applet Instance Deletion): S.ADEL may perform OP.DELETE_APPLET upon an O.APPLET only if,**
  - **(1) S.ADEL is currently selected,**
  - **(2) there is no instance in the context of O.APPLET that is active in any logical channel and**
  - **(3) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance distinct from O.APPLET, or O.JAVAOBJECT is reachable from a package P, or ([R7], §8.5) O.JAVAOBJECT is remote reachable.**
- **R.JAVA.15 ([R7], §11.3.4.1, Multiple Applet Instance Deletion): S.ADEL may perform OP.DELETE_APPLET upon several O.APPLET only if,**
  - **(1) S.ADEL is currently selected,**
  - **(2) there is no instance of any of the O.APPLET being deleted that is active in any logical channel and**
  - **(3) there is no O.JAVAOBJECT owned by any of the O.APPLET being deleted such that either O.JAVAOBJECT is reachable from an applet instance distinct from any of those O.APPLET, or O.JAVAOBJECT is reachable from a package P, or ([R7], §8.5) O.JAVAOBJECT is remote reachable.**
- **R.JAVA.16 ([R7], §11.3.4.2, Applet/Library Package Deletion): S.ADEL may perform OP.DELETE_PCKG upon an O.CODE_PKG only if,**
  - **(1) S.ADEL is currently selected,**

    - **(2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PKG that is an instance of a class that belongs to O.CODE_PKG, exists on the card and**
    - **(3) there is no resident package on the card that depends on O.CODE_PKG.**
  - **R.JAVA.17 ([R7], §11.3.4.3, Applet Package and Contained Instances Deletion): S.ADEL may perform OP.DELETE_PCKG_APPLET upon an O.CODE_PKG only if,**
    - **(1) S.ADEL is currently selected,**
    - **(2) no reachable O.JAVAOBJECT, from a package distinct from O.CODE_PKG, which is an instance of a class that belongs to O.CODE_PKG exists on the card,**
    - **(3) there is no package loaded on the card that depends on O.CODE_PKG, and**
    - **(4) for every O.APPLET of those being deleted it holds that: (i) there is no instance in the context of O.APPLET that is active in any logical channel and (ii) there is no O.JAVAOBJECT owned by O.APPLET such that either O.JAVAOBJECT is reachable from an applet instance not being deleted, or O.JAVAOBJECT is reachable from a package not being deleted, or ([R7], §8.5) O.JAVAOBJECT is remote reachable**.

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 [Editorially Refined] The TSF shall explicitly deny access of **any subject but S.ADEL to O.CODE_PKG or O.APPLET for the purpose of deleting them from the card**.

Application Note:

FDP_ACF.1.2/ADEL:

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

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 packages when one of the deletion operations in FDP_ACC.2.1/ADEL is performed on them**.

Application Note:

Deleted freed resources (both code and data) may be reused, depending on the way they were deleted (logically or physically). Requirements on de-allocation during applet/package deletion are described in [R7], §11.3.4.1, §11.3.4.2 and §11.3.4.3.

FMT_MSA.1/ADEL Management of security attributes

FMT_MSA.1.1/ADEL The TSF shall enforce the **ADEL access control SFP** to restrict the ability to **modify** the security attributes **Registered Applets and Resident Packages** to **the Java Card RE**.

FMT_MSA.3/ADEL Static attribute initialisation

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

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

FMT_SMF.1/ADEL Specification of Management Functions

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

Application Note:

The modification of the Active Applets security attribute should be performed in accordance with the rules given in [R7], §4.

FMT_SMR.1/ADEL Security roles

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

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

FPT_FLS.1/ADEL Failure with preservation of secure state

FPT_FLS.1.1/ADEL The TSF shall preserve a secure state when the following types of failures occur: **the applet deletion manager fails to delete a package/applet as described in [R7], §11.3.4**.

Application Note:

- The TOE may provide additional feedback information to the card manager in case of a potential security violation (see FAU_ARP.1).
- The Package/applet instance deletion must be atomic. The "secure state" referred to in the requirement must comply with Java Card specification ([R7], §11.3.4.)

8.1.4 RMIG Security Functional Requirements

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

FDP_ACC.2/JCRMI Complete access control

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

Refinement:

The operations involved in this policy are:

- o OP.GET_ROR,
- o OP.INVOKE.

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

FDP_ACF.1/JCRMI Security attribute based access control

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

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

.

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

- o **R.JAVA.18: S.CAD may perform OP.GET_ROR upon O.APPLET only if O.APPLET is the currently selected applet, and there exists an O.RMI_SERVICE with a registered initial reference to an O.REMOTE_OBJ that is owned by O.APPLET.**
- o **R.JAVA.19: S.JCRE may perform OP.INVOKE upon O.RMI_SERVICE, O.ROR and O.REMOTE_MTHD only if O.ROR is valid (as defined in [R7], §8.5) and it belongs to the Returned References of O.RMI_SERVICE, and if the Identifier of O.REMOTE_MTHD**

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

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

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

Application Note:

FDP_ACF.1.2/JCRMI:

- The validity of a remote object reference is specified as a lifetime characterization. The security attributes involved in the rules for determining valid remote object references are the Returned References of the O.RMI_SERVICE and the Active Applets (see FMT_REV.1.1/JCRMI and FMT_REV.1.2/JCRMI). The precise mechanism by which a remote method is invoked on a remote object is defined in detail in ([R7], §8.5.2 and [R6]).
- Note that the owner of an O.RMI_SERVICE is the applet instance that created the object. The attribute Returned References lists the remote object references that have been sent to the S.CAD during the applet selection session. This attribute is implementation dependent.

FDP_IFC.1/JCRMI Subset information flow control

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

Application Note:

FDP_IFC.1.1/JCRMI:

- Array parameters of remote method invocations must be allocated on the card as global arrays objects. References to global arrays cannot be stored in class variables, instance variables or array components. The control of the flow of that kind of information has already been specified in FDP_IFC.1.1/JCVM.
- A remote object reference descriptor is sent from the card to the CAD either as the result of a successful applet selection command ([R7], §8.4.1), and in this case it describes, if any, the initial remote object reference of the selected applet; or as the result of a remote method invocation ([R7],§8.3.5.1).

FDP_IFF.1/JCRMI Simple security attributes

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

Subjects/Information	Security attributes
I.RORD	ExportedInfo

.

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

OP.RET_RORD(S.JCRE, S.CAD, I.RORD) is permitted only if the attribute ExportedInfo of I.RORD has the value "true" ([R7], §8.5).

FDP_IFF.1.3/JCRMI The TSF shall enforce the **none**.

FDP_IFF.1.4/JCRMI The TSF shall explicitly authorise an information flow based on the following rules: **none**.

FDP_IFF.1.5/JCRMI The TSF shall explicitly deny an information flow based on the following rules: **the rules describing the communication protocol used by the CAD and the card for transmitting a new package, see chapter 9.3.9 [R9]**.

Application Note:

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

FMT_MSA.1/EXPORT Management of security attributes

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

Application Note:

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

FMT_MSA.1/REM_REFS Management of security attributes

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

FMT_MSA.3/JCRMI Static attribute initialisation

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

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

Application Note:

FMT_MSA.3.1/JCRMI:

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

FMT_MSA.3.2/JCRMI:

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

FMT_REV.1/JCRMI Revocation

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

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

Application Note:

The rules are described in [R7], §8.5

FMT_SMF.1/JCRMI Specification of Management Functions

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

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

FMT_SMR.1/JCRMI Security roles

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

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

Application Note:

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

8.1.5 ODELG Security Functional Requirements

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

FDP_RIP.1/ODEL Subset residual information protection

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

Application Note:

- Freed data resources resulting from the invocation of the method javacard.framework.JCSystem.requestObjectDeletion() may be reused. Requirements on de-allocation after the invocation of the method are described in [R6].
- There is no conflict with FDP_ROL.1 here because of the bounds on the rollback mechanism: the execution of requestObjectDeletion() is not in the scope of the rollback because it must be performed in between APDU command processing, and therefore no transaction can be in progress.

FPT_FLS.1/ODEL Failure with preservation of secure state

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

Application Note:

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

8.1.6 CarG Security Functional Requirements

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

FCO_NRO.2/CM Enforced proof of origin

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

FCO_NRO.2.2/CM [Editorially Refined] The TSF shall be able to relate the **identity** of the originator of the information, and the **application package contained in** the information to which the evidence applies.

FCO_NRO.2.3/CM The TSF shall provide a capability to verify the evidence of origin of information to **recipient** given **immediate verification**.

Application Note:

FCO_NRO.2.1/CM:

- Upon reception of a new application package for installation, the card manager shall first check that it actually comes from the verification authority. The verification authority is the entity responsible for bytecode verification.

FCO_NRO.2.3/CM:

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

FDP_IFC.2/CM Complete information flow control

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

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

Application Note:

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

FDP_IFF.1/CM Simple security attributes

FDP_IFF.1.1/CM The TSF shall enforce the **PACKAGE LOADING information flow control SFP** based on the following types of subject and information security attributes: **LoadFile, Dap**.

FDP_IFF.1.2/CM The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: **the rules describing the communication protocol used by the CAD and the card for transmitting a new package, see chapter 9.3.9 [R9]**.

FDP_IFF.1.3/CM The TSF shall enforce the **none**.

FDP_IFF.1.4/CM The TSF shall explicitly authorise an information flow based on the following rules: **none**.

FDP_IFF.1.5/CM The TSF shall explicitly deny an information flow based on the following rules: **the rules describing the communication protocol used by the CAD and the card for transmitting a new package, see chapter 9.3.9 [R9]**.

Application Note:

FDP_IFF.1.1/CM:

- The security attributes used to enforce the PACKAGE LOADING SFP are implementation dependent. More precisely, they depend on the communication protocol enforced between the CAD and the card. For instance, some of the attributes that can be used are: (1) the keys used by the subjects to encrypt/decrypt their messages; (2) the number of pieces the application package has been split into in order to be sent to the card; (3) the ordinal of each piece in the decomposition of the package, etc. See for example Appendix D of [R12].

FDP_IFF.1.2/CM:

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

FDP_UIT.1/CM Data exchange integrity

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

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

Application Note:

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

FIA_UID.1/CM Timing of identification

FIA_UID.1.1/CM The TSF shall allow **Execution of Card Manager** on behalf of the user to be performed before the user is identified.

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

Application Note:

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

FMT_MSA.1/CM Management of security attributes

FMT_MSA.1.1/CM The TSF shall enforce the **PACKAGE LOADING information flow control SFP** to restrict the ability to **modify** the security attributes **AS.KEYSET_VERSION, AS.KEYSET_VALUE, Default SELECTED Privileges, AS.CMLIFECYC** to **CARD_MANAGER**.

FMT_MSA.3/CM Static attribute initialisation

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

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

FMT_SMF.1/CM Specification of Management Functions

FMT_SMF.1.1/CM The TSF shall be capable of performing the following management functions: **Modify the following security attributes: AS.KEYSET_VERSION, AS.KEYSET_VALUE, Default SELECTED Privileges, AS.CMLIFECYC**.

FMT_SMR.1/CM Security roles

FMT_SMR.1.1/CM The TSF shall maintain the roles **Card manager**.

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

FTP_ITC.1/CM Inter-TSF trusted channel

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

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

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

Application Note:

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

8.1.6.1 Additional Security Functional Requirements for CM

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**.

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

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

FCO_NRO.2/CM_DAP Enforced proof of origin

FCO_NRO.2.1/CM_DAP The TSF shall enforce the generation of evidence of origin for transmitted **Loadfile** at all times.

FCO_NRO.2.2/CM_DAP The TSF shall be able to relate the **AS.KEYSET_VALUE** of the originator of the information, and the **CAP file components** of the information to which the evidence applies.

FCO_NRO.2.3/CM_DAP The TSF shall provide a capability to verify the evidence of origin of information to **recipient** given **during CAP file loading**.

FIA_AFL.1/CM Authentication failure handling

FIA_AFL.1.1/CM The TSF shall detect when **1** unsuccessful authentication attempts occur related to **U.Card_Issuer authentication**.

FIA_AFL.1.2/CM When the defined number of unsuccessful authentication attempts has been **met and surpassed**, the TSF shall **slow down exponentially the next authentication**.

FIA_UAU.1/CM Timing of authentication

FIA_UAU.1.1/CM The TSF shall allow **Get_Data, Initialize_Update, Select** on behalf of the user to be performed before the user is authenticated.

FIA_UAU.1.2/CM 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/CardIssuer Single-use authentication mechanisms

FIA_UAU.4.1/CardIssuer The TSF shall prevent reuse of authentication data related to **the Card Issuer authentication mechanism**.

FIA_UAU.7/CardIssuer Protected authentication feedback

FIA_UAU.7.1/CardIssuer The TSF shall provide only **the result of the authentication (NOK), the key set version, Secure channel identifier and the card random and the card cryptogram** to the user while the authentication is in progress.

FPR_UNO.1/Key_CM Unobservability

FPR_UNO.1.1/Key_CM The TSF shall ensure that **all subjects** are unable to observe the operation **OP.IMPORT_KEY** on **Key** by **D.JCS_KEYS**.

FPT_TDC.1/CM Inter-TSF basic TSF data consistency

FPT_TDC.1.1/CM The TSF shall provide the capability to consistently interpret **AS.KEYSET_VALUE, Packages** when shared between the TSF and another trusted IT product.

FPT_TDC.1.2/CM The TSF shall use **the PUT KEY data format** when interpreting the TSF data from another trusted IT product.

FMT_SMR.2/CM Restrictions on security roles

FMT_SMR.2.1/CM The TSF shall maintain the roles: **see below**.

FMT_SMR.2.2/CM The TSF shall be able to associate users with roles.

FMT_SMR.2.3/CM The TSF shall ensure that the conditions **see details below:**

Roles	Condition for this role
R.personaliser	Successful authentication (Card Issuer) using a key set of the Card Manager or Security Domain associates with CM life cycle phase from OP_READY to SECURED
R.Card_Manager	Successful authentication (of Card Issuer) using its key set, with CM life cycle phase from OP_READY to SECURED
R.Security_Domain	Successful authentication (of application provider) using its key set, with CM life cycle phase different from locked
R.Use_API	Successful identification (of Applet), with Applet life cycle phase after SELECTABLE
R.Applet_privilege	have the privilege to modify CM life cycle, ATR, and also Global Pin

are satisfied.

FCS_COP.1/CM Cryptographic operation

FCS_COP.1.1/CM The TSF shall perform **see table below** in accordance with a specified cryptographic algorithm **see table below** and cryptographic key sizes **see table below** that meet the following:

Cryptographic operation	Algorithm	Key length	Standard
TOE authentication key ISK/KMC	SCP02	112 bits	GP 2.1.1
TOE authentication key ISK/KMC	SCP03	128/192/256 bits	GP 2.1.1
SCP02 - signature, verification of signature, encryption and decryption	TDES	112 bits	SCP02 – GP 2.1.1
SCP03 - signature, verification of signature, encryption and decryption	AES	128/192/256 bits	SCP03 – GP 2.1.1

Cryptographic operation	Algorithm	Key length	Standard
SCPF3 - signature, verification of signature, encryption and decryption	AES	128 bits	Proprietary

.

8.1.6.2 Additional Security Functional Requirements for Resident application

FDP_ACC.2/PP Complete access control

FDP_ACC.2.1/PP The TSF shall enforce the **See below** on **See below** and all operations among subjects and objects covered by the SFP

Access Control	
Prepersonalisation Access Control	S.Resident application and for all obj
Patch & Locks Loading Access Control	S.TOE and for all objects

.

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

Application note:
This SFR enforces the access control for the patch and locks loading and the ISK loading.

FDP_ACF.1/PP Security attribute based access control

FDP_ACF.1.1/PP The TSF shall enforce the **See below** to objects based on the following:

Access Control	
Prepersonalisation Access Control	AS_AUTH_MSK_STATUS
Patch & Locks Loading Access Control	AS_AUTH_MSK_STATUS

.

FDP_ACF.1.2/PP The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: **AS.AUTH_MSK_STATUS=TRUE**.

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

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

FDP_UCT.1/PP Basic data exchange confidentiality

FDP_UCT.1.1/PP The TSF shall enforce the **Prepersonalisation access control and Patch and Locks loading access control** to **receive** user data in a manner protected from unauthorised disclosure.

FDP_ITC.1/PP Import of user data without security attributes

FDP_ITC.1.1/PP The TSF shall enforce the **Prepersonalisation access control and Patch and Locks loading access control** when importing user data, controlled under the SFP, from outside of the TOE.

FDP_ITC.1.2/PP The TSF shall ignore any security attributes associated with the user data when imported from outside the TOE.

FDP_ITC.1.3/PP The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TOE: **none**.

FIA_AFL.1/PP Authentication failure handling

FIA_AFL.1.1/PP The TSF shall detect when **3** unsuccessful authentication attempts occur related to **U.Card_manufacturer authentication**.

FIA_AFL.1.2/PP When the defined number of unsuccessful authentication attempts has been **met**, the TSF shall **always return an error**.

FIA_UAU.1/PP Timing of authentication

FIA_UAU.1.1/PP The TSF shall allow **INITIALIZE AUTHENTICATION PROCESS, GET DATA, MANAGE CHANNEL, SELECT APPLET** on behalf of the user to be performed before the user is authenticated.

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

FIA_UID.1/PP Timing of identification

FIA_UID.1.1/PP The TSF shall allow **INITIALIZE AUTHENTICATION PROCESS, GET DATA, MANAGE CHANNEL, SELECT APPLET** on behalf of the user to be performed before the user is identified.

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

FMT_MSA.1/PP Management of security attributes

FMT_MSA.1.1/PP The TSF shall enforce the **Prepersonalisation access control** to restrict the ability to **modify** the security attributes **AS.AUTH_MSK_STATUS** to **R.Prepersonaliser**.

FMT_SMF.1/PP Specification of Management Functions

FMT_SMF.1.1/PP The TSF shall be capable of performing the following management functions: **modify security attributes**.

FIA_ATD.1/CardManu User attribute definition

FIA_ATD.1.1/CardManu The TSF shall maintain the following list of security attributes belonging to individual users: **AS.AUTH_MSK_STATUS**.

FIA_UAU.4/CardManu Single-use authentication mechanisms

FIA_UAU.4.1/CardManu The TSF shall prevent reuse of authentication data related to **the Card Manufacturer authentication mechanism**.

FIA_UAU.7/CardManu Protected authentication feedback

FIA_UAU.7.1/CardManu The TSF shall provide only **the result of the authentication (NOK) and the random** to the user while the authentication is in progress.

FMT_MOF.1/PP Management of security functions behaviour

FMT_MOF.1.1/PP The TSF shall restrict the ability to **see below** the functions **see below** to:

	Functions	Role
Disable	INITIALIZE AUTHENTICATION PROCESS, EXTERNAL AUTHENTICATE, LOAD STRUCTURE, INSTALL, LOAD SECURE, LOAD APPLET, GET DATA	R.Prepersonaliser
Modify	Self tests described in FPT_TST.1	R.Prepersonaliser
Modify the behaviour	All functions	R.Developer

Application note:

The first operation ensures the irreversible locking of the patch and locks loading features once in OP_READY, after pre production state. Once in OP_READY state, those APDU can not be used.
The second operation described the product configuration regarding self tests, as described in AGD_PRE, chapter 8 [R39].

The last operation permits the loading of patch and locks during phase 5.

FMT_SMR.2/PP Restrictions on security roles

FMT_SMR.2.1/PP The TSF shall maintain the roles: **R.Prepersonaliser and R.Developer**.

FMT_SMR.2.2/PP The TSF shall be able to associate users with roles.

FMT_SMR.2.3/PP The TSF shall ensure that the conditions **see refinement below** are satisfied.

Refinement:

Roles	Condition for this role
R.Prepersonaliser	Successful authentication (of Card Manufacturer) using MSK and card still in prepersonalisation state, in phase 4-5.
R.Developer	Succesful authentication (of TOE developer) using LSK in phase 4-5

FMT_MSA.3/PP Static attribute initialisation

FMT_MSA.3.1/PP The TSF shall enforce the **Prepersonalisation access control** to provide **same rights by** default values for security attributes that are used to enforce the SFP.

FMT_MSA.3.2/PP 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.

FCS_COP.1/PP Cryptographic operation

FCS_COP.1.1/PP The TSF shall perform **see table below** in accordance with a specified cryptographic algorithm **see table below** and cryptographic key sizes **see table below** that meet the following:

Cryptographic operation	Algorithm	Key length	Standard
Decryption (MSK) and signature verification	DES	112 bits	FIPS-PUB 46-3 (ANSI X3.92), FIPS PUB 81 or ISO/IEC 9797, Data integrity mechanism
Card Manufacturer authentication (MSK)	DES	112 bits	FIPS PUB 197
Card Manufacturer authentication (MSK)	AES	128, 192 and 256 bits	FIPS-PUB 46-3 (ANSI X3.92), FIPS PUB 81 or ISO/IEC 9797, Data integrity mechanism
Decryption (of patch and locks ciphered with LSK) and signature verification	TDES	112 bits	FIPS-PUB 46-3 (ANSI X3.92), FIPS PUB 81 or ISO/IEC 9797, Data integrity mechanism
TOE authentication key ISK/KMC	TDES	112 bits	FIPS PUB 197

.

FCS_CKM.4/PP Cryptographic key destruction

FCS_CKM.4.1/PP The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method **Key is set to NULL** that meets the following: **no**.

FDP_UIT.1/PP Data exchange integrity

FDP_UIT.1.1/PP The TSF shall enforce the **Patch and locks and Prepersonalisation loading access control SFP** to **receive** user data in a manner protected from **modification** errors.

FDP_UIT.1.2/PP [Editorially Refined] The TSF shall be able to determine on receipt of user data, whether **modification of some of the pieces of the application sent by the TOE developer and Card Manufacturer** has occurred.

FCS_CKM.1/PP Cryptographic key generation

FCS_CKM.1.1/PP The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm **see table below** and specified cryptographic key sizes **see table below** that meet the following: **see table below:**

Cryptographic key generation algorithm	Cryptographic key size	List of standards
TOE's MSK derived from the MSK loaded in phase 1, using SHA-256	16, 24 and 32 bytes	None

FTP_ITC.1/PP Inter-TSF trusted channel

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

FTP_ITC.1.2/PP [Editorially Refined] The TSF shall permit **the TOE Developer and Card Manufacturer** to initiate communication via the trusted channel.

FTP_ITC.1.3/PP The TSF shall initiate communication via the trusted channel for **loading the patch code, locks and ISK on the card**.

FAU_STG.2 Guarantees of audit data availability

FAU_STG.2.1 The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

FAU_STG.2.2 The TSF shall be able to **prevent** unauthorized modifications to the stored audit records in the audit trail.

FAU_STG.2.3 The TSF shall ensure that **Patch code identification** stored audit records will be maintained when the following conditions occur: **audit storage exhaustion, failure and attack**.

8.1.6.3 Additional Security Functional Requirements for SmartCard Platform

FPT_PHP.3/SCP Resistance to physical attack

FPT_PHP.3.1/SCP The TSF shall resist **physical manipulation and physical probing** to the **all TOE components implementing the TSF** by responding automatically such that the SFRs are always enforced.

Application Note:

The physical manipulation and physical probing include: changing operational conditions every times: the frequency of the external clock, power supply, and temperature

FPT_FLS.1/SCP Failure with preservation of secure state

FPT_FLS.1.1/SCP The TSF shall preserve a secure state when the following types of failures occur: **cf FAU_ARP.1**.

FPT_RCV.3/SCP Automated recovery without undue loss

FPT_RCV.3.1/SCP When automated recovery from **none** 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/SCP For **all cases**, the TSF shall ensure the return of the TOE to a secure state using automated procedures.

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

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

FPT_RCV.4/SCP Function recovery

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

FRU_FLT.1/SCP Degraded fault tolerance

FRU_FLT.1.1/SCP The TSF shall ensure the operation of **Fault tolerance** when the following failures occur: **Lack of EEPROM**.

Application Note:

The TOE implements a mechanism to detect a problem of EEPROM. During the
life of the TOE, the Transaction area reduces its size to skip damaged EEPROM bytes. During the writing or erasing operations, up to 3 maximum attempts to get successful programming are done.
Otherwise the EXCEPTION_EEPROM_ERROR is raised.

FPR_UNO.1/USE_KEY Unobservability

FPR_UNO.1.1/USE_KEY The TSF shall ensure that **all subjects** are unable to observe the operation **use** on **key** by **D.JCS_KEYS**.

FCS_RNG.1/SCP Random Number Generation

FCS_RNG.1.1/SCP The TSF shall provide a **deterministic hybrid** random number generator that implements: **none**.

FCS_RNG.1.2/SCP The TSF shall provide random numbers that meet **RGS_B1 [R31]**.

8.1.6.4 Additional Security Functional Requirements for the applets

FIA_AFL.1/PIN Authentication failure handling

FIA_AFL.1.1/PIN The TSF shall detect when **an administrator configurable positive integer within from 1 to 127 for OwnerPIN** unsuccessful authentication attempts occur related to **any user authentication using a PIN**.

FIA_AFL.1.2/PIN When the defined number of unsuccessful authentication attempts has been **met**, the TSF shall **block the PIN**.

FMT_MTD.2/GP_PIN Management of limits on TSF data

FMT_MTD.2.1/GP_PIN The TSF shall restrict the specification of the limits for **D.NB_REMAINTRYGLB, GlobalPIN** to **R.Card_Manager**.

FMT_MTD.2.2/GP_PIN The TSF shall take the following actions, if the TSF data are at, or exceed, the indicated limits: **block D.PIN**.

FPR_UNO.1/Applet Unobservability

FPR_UNO.1.1/Applet The TSF shall ensure that **S.APPLET** are unable to observe the operation **Comparison** on **two bytes arrays** by **D.ARRAY**.

FMT_MTD.1/PIN Management of TSF data

FMT_MTD.1.1/PIN The TSF shall restrict the ability to **change_default, query and modify** the **OwnerPIN** to **applet itself**.

FIA_AFL.1/GP_PIN Authentication failure handling

FIA_AFL.1.1/GP_PIN The TSF shall detect when **an administrator configurable positive integer within 3 to 15** unsuccessful authentication attempts occur related to **any user authentication using a Global PIN**.

FIA_AFL.1.2/GP_PIN When the defined number of unsuccessful authentication attempts has been **met**, the TSF shall **block the PIN**.

8.1.6.5 Additional Security Functional Requirements for BIO

FIA_AFL.1/PIN_BIO Authentication failure handling

FIA_AFL.1.1/PIN_BIO The TSF shall detect when **an administrator configurable positive integer within user defined maximum from 1 to 254 for BIOMETRIC_DATA** unsuccessful authentication attempts occur related to **any user authentication using MOC**.

FIA_AFL.1.2/PIN_BIO When the defined number of unsuccessful authentication attempts has been **met**, the TSF shall **block the MOC**.

FMT_MTD.1/PIN_BIO Management of TSF data

FMT_MTD.1.1/PIN_BIO The TSF shall restrict the ability to **change_default, query and modify** the **BIOMETRIC_DATA** to **applet itself**.

8.1.6.6 Additional Security Functional Requirements for Runtime Verification

8.1.6.6.1 Stack Control

FDP_ACC.2/RV_Stack Complete access control

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

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

FDP_ACF.1/RV_Stack Security attribute based access control

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

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

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

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

FMT_MSA.1/RV_Stack Management of security attributes

FMT_MSA.1.1/RV_Stack The TSF shall enforce the **[assignment: *access control SFP(s), information flow control SFP(s)*]** to restrict the ability to **[selection: *change_default, query, modify, delete, [assignment: other operations]*]** the security attributes **[assignment: *list of security attributes*]** to **[assignment: *the authorised identified roles*]**.

FMT_MSA.2/RV_Stack Secure security attributes

FMT_MSA.2.1/RV_Stack The TSF shall ensure that only secure values are accepted for **[assignment: *list of security attributes*]**.

FMT_MSA.3/RV_Stack Static attribute initialisation

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

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

FMT_SMF.1/RV_Stack Specification of Management Functions

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

8.1.6.6.2 Heap Access

FDP_ACC.2/RV_Heap Complete access control

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

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

FDP_ACF.1/RV_Heap Security attribute based access control

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

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

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

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

FMT_MSA.1/RV_Heap Management of security attributes

FMT_MSA.1.1/RV_Heap The TSF shall enforce the **[assignment: *access control SFP(s), information flow control SFP(s)*]** to restrict the ability to **[selection: *change_default, query, modify, delete, [assignment: other operations]*]** the security attributes **[assignment: *list of security attributes*]** to **[assignment: *the authorised identified roles*]**.

FMT_MSA.2/RV_Heap Secure security attributes

FMT_MSA.2.1/RV_Heap The TSF shall ensure that only secure values are accepted for **[assignment: *list of security attributes*]**.

FMT_MSA.3/RV_Heap Static attribute initialisation

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

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

FMT_SMF.1/RV_Heap Specification of Management Functions

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

8.1.6.6.3 Transient Control

FDP_ACC.2/RV_Transient Complete access control

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

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

FDP_ACF.1/RV_Transient Security attribute based access control

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

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

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

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

FMT_MSA.1/RV_Transient Management of security attributes

FMT_MSA.1.1/RV_Transient The TSF shall enforce the **[assignment: *access control SFP(s), information flow control SFP(s)*]** to restrict the ability to **[selection: *change_default, query, modify, delete, [assignment: other operations]*]** the security attributes **[assignment: *list of security attributes*]** to **[assignment: *the authorised identified roles*]**.

FMT_MSA.2/RV_Transient Secure security attributes

FMT_MSA.2.1/RV_Transient The TSF shall ensure that only secure values are accepted for **[assignment: *list of security attributes*]**.

FMT_MSA.3/RV_Transient Static attribute initialisation

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

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

FMT_SMF.1/RV_Transient Specification of Management Functions

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

9 TOE Summary Specification

9.1 TOE Summary Specification

SF_ATOMIC_TRANSACTION

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 TOE behaves as if none of them had been attempted. The transaction mechanism is used for updating internal TSF data as well as for performing different functions of the TOE, like installing a new package on the card. This TSF is also available for applet instances through the javacard.framework.JCSystem, javacard.framework.Util and javacardx.framework.util.ArrayLogic classes. The first class provides the applet instances with methods for starting, aborting and committing a sequence of modifications of the persistent memory. The other classes provide methods for atomically copying arrays. This TSF ensures that the following data is never updated conditionally:

- The validated flag of the PINs
- The validated flag of the BIO template
- The reason code of the CardException and CardRuntimeException
- Transient objects
- Global arrays, like the APDU buffer and the buffer that the applet instances use to store installation data
- Any intermediate result state in the implementation instance of the Checksum, Signature, Cipher, and Message Digest classes of the JavaCard API.

This TSF also performs the actions necessary to roll back to a safe state upon interruption of the following procedures, for example because of a card withdrawal or an unexpected fatal error:

- Loading and linking of a package
- Installing a new applet instance
- Deleting a package
- Deleting an applet instance
- Collecting unreachable objects
- Reading from and writing to a static field, instance field or array position
- Populating, updating or clearing a cryptographic key
- Modifying a PIN value

Finally, this TSF ensures that no transaction is in progress when a method of an applet instance is invoked for installing, deselecting, selecting or processing an APDU sent to the applet instance. Concerning memory limitations on the transaction journal, this TSF guarantees that an exception is thrown when the maximal capacity is reached. The TSF preserves a secure state when such limit is reached. Atomic Transactions are detailed in the chapter Atomicity and Transactions of the [R7] and in the documentation associated to the JCSystem class in the [R6].

SF_CARD_CONTENT_MANAGEMENT

This TSF ensures the following functionalities:

- Loading (Section 9.3.5 of [R12]): This function allows the addition of code to mutable persistent memory in the card. During card content loading, this TSF checks that the required packages are already installed on the card. If one of the required packages does not exist, or if the version installed on the card is not binary compatible with the version required, then the loading of the package is rejected. Loading is also rejected if the version of the CAP format of the package is newer than the one supported by the TOE. If any of those checks fails, a suitable error message is returned to the CAD.

- Installation (Section 9.3.6 of [R12]): This function allows the Installer to create an instance of a previously loaded Applet subclass and make it selectable. In order to do this, the install() method of the Applet subclass is invoked using the context of that new instance as the currently active context. If this method returns with an exception, the exception is trapped and the smart card rolls back to the state before starting the installation procedure.
- Deletion (Section 9.5 of [R12]): This function allows the Applet Deletion Manager to remove the code of a package from the card, or to definitely deactivate an applet instance, so that it becomes no longer selectable. This TSF performs physical removal of those packages and applet data stored in NVRAM, while only logical removal is performed for packages in ROM. This TSF checks that the package or applet actually exists, and that no other package or applet depends on it for its execution. In this case, the entry of the package or applet is removed from the registry, and all the objects on which they depend are garbage collected. Otherwise, a suitable error is returned to the CAD. The deletion of the Applet Deletion Manager, the Installer or any of the packages required for implementing the Java Card platform Application Programming Interface (Java Card API) is not allowed.
- Extradition (Section 9.4.1 of): This function allows the Installer to associate load files or applet instances to a Security Domain different than their currently associated Security Domain. It is also used to associate a Security Domain to another Security Domain or to itself thus creating Security Domains hierarchies. If this method returns with an exception, the exception is trapped and the smart card rolls back to the state before starting the extradition procedure.
- Registry update (Section 9.4.2 of): This function allows the Installer to populate, modify or delete elements of the Registry entry of applet instances. If this method returns with an exception, the exception is trapped and the smart card rolls back to the state before starting the extradition procedure.

SF_CARD_MANAGEMENT_ENVIRONMENT

This TSF is in charge of initializing and managing the internal data structures of the Card Manager. During the initialization phase of the card, this TSF creates the Installer and the Applet Deletion Manager and initializes their internal data structures. The internal data structures of the Card Manager includes the Package and Applet Registries, which respectively contains the currently loaded packages and the currently installed applet instances, together with their associated AIDs. This TSF is also in charge of dispatching the APDU commands to the applets instances installed on the card and keeping traces of which are the currently active ones. It therefore handles sensitive TSF data of other security functions, like the Firewall or the Remote Access Control function.

SF_CARDHOLDER_VERIFICATION

This TSF enables applet instances to authenticate the sender of a request as the true cardholder. Applet instances have access to these services through the OwnerPIN class. Cardholder authentication is performed using the following security attributes:

- A secret enabling to authenticate the cardholder
- The maximum number of consecutive unsuccessful comparison attempts that are admitted
- A counter of the number of consecutive unsuccessful comparison attempts that have been performed so far

- The current life cycle state of the secret (reference value). This state is always updated, even if the modification is in the scope of an open transaction. Each time an attempt is made to compare a value to the reference value, and prior to the comparison being actually performed, if the reference is blocked, then the comparison fails and the reference value is not accessed. Otherwise, the try counter is decremented by one. This operation is always performed, even if it is in the scope of an open transaction. If the comparison is successful, then the try counter is reset to the try limit. When the try counter reaches zero, the reference enters into a blocked state, and cannot be used until it is unblocked. Cardholder Verification Method services are implemented to resist to environmental stress and glitches and include measures for preventing information leakage through covert channels. In particular, unsuccessful authentication attempts consume the same power and execution time than successful ones. The Cardmanager uses the class OwnerPin to provide the services to the Applet that want benefit of the Shared GP_PIN.

SF_CLEARING_OF_SENSITIVE_INFORMATION

This TSF clears all the data containers that hold sensitive information when that information is no longer used. This includes:

- The contents of the memory blocks allocated for storing class instances, arrays, static field images and local variables, before allocating a fresh block
- The objects reclaimed by the Java Card VM garbage collector
- The code of the deleted packages
- The objects accessible from a deleted applet instance
- All the information contained in the packages that is not necessary for executing the code of the applets, like the Descriptor Component, the Reference Location Component and the Constant Pool of the CAP files
- The contents of the APDU buffer after processing an APDU command
- The content of the bArray argument of the Applet.install method after a new applet instance is installed
- The content of CLEAR ON DESELECT transient objects owned by an applet instance that has been deselected when no other applets from the same package are active on the card
- The content of all transient objects after a card reset
- The reason code contained in the instances of a CardException or CardRuntimeException classes after a card reset
- The validated flag of the PINs after a card reset
- The validated flag of the BIO templates after a card reset
- The contents of the cryptographic buffer after performing cryptographic operations
- The content of the input parameters of a remote method invocation after returning the response to the terminal

Application Note:

This function is in charge of clearing the information contained in the objects that are no longer accessible from the installed packages and applet instances. Clearing is performed on demand of an applet instance through the JCSystem.requestObjectDeletion() method.

SF_DAP_VERIFICATION

An Application Provider may require that its Application code to be loaded on the card is checked for integrity and authenticity. The DAP Verification privilege of the Application Provider's Security

Domain detailed in Section 9.2.1 of provides this service on behalf of an Application Provider. A Controlling Authority may require that all Application code to be loaded onto the card shall be checked for integrity and authenticity. The Mandated DAP Verification privilege of the Controlling Authority's Security Domain detailed in Section 9.2.1 of provides this service on behalf of the Controlling Authority. The keys and algorithms to be used for DAP Verification or Mandated DAP Verification are implicitly known by the corresponding Security Domain.

SF_DATA_COHERENCY

As coherency of data should be maintained, and as power is provided by the CAD and might be stopped at all moment (by tearing or attacks), a transaction mechanism is provided. When updating data, before writing the new ones, the old ones are saved in a specific memory area. If a failure appears, at the next start-up, if old data are valid in the transaction area, the system restores them for staying in a coherent state.

SF_DATA_INTEGRITY

Some of the data in non volatile memory can be protected. Keys, PIN, BIO templates package and patch code are protected with integrity value. When reading and writing operation are, the integrity value is checked and maintained valid. In case of incoherency, an exception is raise to prevent the bad use of the data. SecureStore is a mean for protecting JavaCard data in integrity.

SF_ENCRYPTION_AND_DECRYPTION

This TSF provides the applet instances with mechanisms for encrypting and decrypting the contents of a byte array.

The ciphering algorithms are available to the applets through the Cipher class of the Java Card API, ISOSecureMessaging class and SecureChannel class. The length of the key to be used for the ciphering operation is defined by the applet instance when the key is generated. Before encrypting or decrypting the byte array, the TSF verifies that the specified key has been previously initialized, and that is in accordance with the specified ciphering algorithm (DES, RSA, etc). The TSF also checks that it has been provided with all the information necessary for the encryption/decryption operation. Once the ciphering operation is performed, the internal TSF data used for the operation like the ICV is cleared. Ciphering operations are implemented to resist to environmental stress and glitches and include measures for preventing information leakage through covert channels.

Mechanisms of encrypting and decrypting for Secure Messaging are available to the applets through the SecureChannel (Global Platform Card 2.2" specification) and ISOSecureMessaging (Proprietary API) classes.

SF_ENTITY_AUTHENTICATION/SECURE_CHANNEL

Off-card entity authentication is achieved by initiating a Secure Channel and provides assurance to the card that it is communicating with an authenticated off-card entity. If any step in the off-card authentication process fails, the process shall be restarted (i.e. new session keys generated). The Secure Channel initiation and off-card entity authentication implies the creation of session keys derived from card static key(s).

SF_EXCEPTION

In case of abnormal event: data unavailable on an allocation, illegal access to a data, the system owns an internal mechanism that allows to stop the code execution and raise an exception.

SF_FIREWALL

This TSF enforces the Firewall security policy and the information flow control policy at runtime. The former policy controls object sharing between different applet instances, and between applet instances and the Java Card RE. The latter policy controls the access to global data containers shared by all applet instances. This TSF is enforced by the Java Card platform Virtual Machine (Java Card VM). During the execution of an applet, the Java Card VM keeps track of the applet instance that is currently performing an action. This information is known as the currently active context. Two kinds of contexts are considered: applet instances contexts and the Java Card RE context, which has special privileges for accessing objects. The TSF makes no difference between instances of applets defined in the same package: all of them belong to the same active context. On the contrary, instances of applets defined in different packages belong to different contexts. Each object belongs to the context that was active when the object was allocated. Initially, when the Java Card VM is launched, the context corresponding to the applet instance selected for execution becomes the first active context. Each time an instance method is invoked on an object, a context switch is performed, and the owner of the object becomes the new active context. On the contrary, the invocation of a static method does not entail a context switch. Before executing a bytecode that accesses an object, the object's owner is checked against the currently active context in order to determine if access is allowed. Access is determined by the Firewall access control rules specified in the chapter Applet Isolation and Object Sharing of the [R7]. Those rules enable controlled sharing of objects through interface methods that the object's owner explicitly exports to other applet instances, and provided that the object's owner explicitly accepts to share it upon request of the method's invoker.

SF_GP_DISPATCHER

While a Security Domain is selected, this function tests for every command, according to the Security Domain life cycle state and the Card life cycle state, if security requirements are needed (if a Secure Channel is required).

SF_HARDWARE_OPERATING

When needed, at each start up or before first use, a self test of each hardware functional module is done, i.e.: DES, RSA, RNG implements a know calculus and checks if the result is correct. When executing, external hardware event can be trigged to prevent attacks or bad use. Temperature, frequency, voltage, light, glitch are considered as abnormal environmental conditions and put the card in frozen state. The TOE shall monitor IC detectors (e.g. out-of-range voltage, temperature, frequency, active shield, memory aging) and shall provide automatic answers to potential security violations through interruption routines that leave the device in a secure state.

The TOE with the IC has detectors of operational conditions. It shall resist to attackers with high-attack potential according to [R36] characterisation, in particular, to leakage attacks, intrusive (e.g. probing, fault injection) and non-intrusive (e.g. SPA, DPA, EMA) attacks, operational conditions manipulation (voltage, clock, temperature, etc) and physical attacks aiming at modification of the IC content or behaviour. To be compliant to related SUN Protection Profile [R5], the off-card verifier is mandatory in this ST; however, this TOE runs some additional verification at execution time. These verifications ensure that: 1. No read accesses are made to Java Card System code, data belonging to another application, data belonging to the Java Card System, 2. No write accesses are made to another application's code, Java Card System code, another application's data Java Card System or API data, 3. No execution of code is done from a method or from a method fragment belonging to another package (including execution on arbitrary data).

SF_KEY_ACCESS

This TSF enforces secure access to all cryptographic keys of the card: RSA keys, DES keys, EC keys, AES keys

SF_KEY_AGREEMENT

This TSF provides the applet instances with a mechanism for supporting key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363].

SF_KEY_DESTRUCTION

This TSF disables the use of a key both logically and physically. When a key is cleared, the internal life cycle of the key container is moved to a state in which no operation is allowed. Applet instances may invoke this TSF through the interfaces declared in the javacard.security package of the Java Card API.

SF_KEY_DISTRIBUTION

This TSF enforces the distribution of all the cryptographic keys of the card using the method specified in that SFR.

SF_KEY_GENERATION

This TSF enforces the creation and/or the oncard generation of all the cryptographic keys of the card using the method specified in that SFR.

SF_KEY_MANAGEMENT

This function enables key sets management (PIN, BIO). It allows creating updating and deleting key sets. It is used to load keys to the card. It also implements verification of Key sets attributes: key lengths, key types... and enforces the loaded keys integrity

SF_MANUFACTURER_AUTHENTICATION

At prepersonalisation phase, manufacturer authentication at the beginning of a communication session is mandatory prior to any relevant data being transferred to the TOE.

SF_MESSAGE_DIGEST

This TSF provides the applet instances with a mechanism for generating an (almost) unique value for a byte array content. That value can be used as a short representative of the information contained in the whole byte array. The hashing algorithms are available to the applets through the MessageDigest class of the Java Card API. Before generating the hash value, the TSF verifies that it has been provided with all the information necessary for the hashing operation. For those algorithms that do not pad the messages, the TSF checks that the information is block aligned before computing its hash value. Message digest generation is implemented to resist to environmental stress and glitches and include measures for preventing information leakage through covert channels.

SF_MEMORY_FAILURE

When using the non volatile memory, in case of a bad writing, internal mechanisms are implemented to prevent an incoherency of the written data. In case of an impossible writing, an exception is raised

SF_PREPERSONALISATION

This function is in charge of pre-initializing the internal data structures, loading the configuration of the card and loading patch code if needed.

SF_RANDOM_NUMBER

This TSF provides to card manager, resident application, applets a mechanism for generating challenges and key values. Random number generators are available to the applets through the

RandomData class of the Java Card API. Off-card entity authentication is achieved through the process of initiating a Secure Channel and provides assurance to the card that it is communicating with an authenticated off-card entity. If any step in the off-card authentication process fails, the process shall be restarted (i.e. new session keys generated). The Secure Channel initiation and off-card entity authentication implies the creation of session keys derived from card static key(s).

SF_RESIDENT_APPLICATION_DISPATCHER

During prepersonalisation phase, this function tests for every command if manufacturer authentication is required.

SF_REMOTE_ACCESS

This TSF enforces the access control to remote objects when the RMI service is used. The Remote objects and its security attributes are created and initialized at the creation of the object.

SF_RUNTIME_VERIFIER

This security functionality ensures the secure processing of information by ensuring the following elements:

- Stack Control
- Heap Control
- Transient Control

Information on the processing is described on the related FDP_ACF.1.

SF_SECURITY_FUNCTIONS_OF_THE_IC

The TOE uses the security functions of the IC. The list of the security function is presented in the ST lite of the IC component

SF_SIGNATURE

This TSF provides the applet instances with a mechanism for generating an electronic signature of a byte array content and verifying an electronic signature contained in a byte array. An electronic signature is made of a hash value of the information to be signed encrypted with a secret key. The verification of the electronic signature includes decrypting the hash value and checking that it actually corresponds to the block of signed bytes.

The signature algorithms are available to the applets through the javacard.Signature class of the Java Card API, ISOSecureMesssaging class and SecureChannel class. The length of the key to be used for the signature is defined by the applet instance when the key is created. Before generating the signature, the TSF verifies that the specified key is suitable for the operation (secret keys for signature generation), that it has been previously initialized, and that is in accordance with the specified signature algorithm (DES, RSA, etc). The TSF also checks that it has been provided with all the information necessary for the signature operation. For those algorithms that do not pad the messages, the TSF checks that the information to be signed is block aligned before performing the signature operation. Once the signature operation is performed, the internal TSF data used for the operation like the ICV is cleared. Signature operations are implemented to resist to environmental stress and glitches and include measures for preventing information leakage through covert channels.

Mechanisms of signature for Secure Messaging are available to the applets through the SecureChannel (Global Platform Card 2.2" specification) and ISOSecureMessaging (Proprietary API) classes. The signature is included in Data Objects.

SF_UNOBSERVABILITY

This function assures that processing based on secure elements of the TOE does not reveal any information on those elements. For example, observation of a PIN verification cannot reveal the PIN value, observation a cryptographic computation cannot give information on the key.

10 Related documents

Ref	Document details
[R1]	"Common Criteria for information Technology Security Evaluation, Part 1: Introduction and general model" July 2009, Version 3.1 revision 3.
[R2]	"Common Criteria for information Technology Security Evaluation, Part 2: Security Functional requirements" July 2009, Version 3.1 revision 3.
[R3]	"Common Criteria for information Technology Security Evaluation, Part 3: Security Assurance requirements" July 2009, Version 3.1 revision 3.
[R4]	“Composite product evaluation for Smart Cards and similar devices” September 2007, Version 1.0, CCDB-2007-09-001.
[R5]	PP SUN Java Card™ System Protection Profile Open Configuration v3.0 May 2012, ANSSI-CC-PP-2010/03_M01
[R6]	"Java Card - API" Application Programming Interfaces, Classic Edition Version 3.01, February 23, 2009, Sun Microsystems.
[R7]	"Java Card – JCRE” Runtime Environment Specification, Classic Edition Version 3.01, February 23, 2009, Sun Microsystems.
[R8]	"Java Card - Virtual Machine Specifications" Classic Edition, Version 3.01 February 23, 2009, Sun Microsystems.
[R9]	Global Platform, Card Specification Version 2.2.1 – January 2011.
[R10]	Global Platform Card, Mapping Guidelines of Existing GP v2.1.1 Implementation on v2.2.1 Version 1.0.1 – January 2011.
[R11]	Global Platform Card, ID Configuration Version 1.0 - December 2011.
[R12]	Global Platform Card Technology, Secure Channel Protocol 03, Card Specification v 2.2 - Amendment D Version 1.1 - September 2009.
[R13]	Global Platform Card Technology, Security Upgrade for Card Content Management, Card Specification v 2.2 – Amendment E Version 0.14 - October 2011.

Ref	Document details
[R14]	"Identification cards - Integrated Circuit(s) Cards with contacts, Part 6: Interindustry data elements for interchange" ISO/IEC 7816-6 (2004)
[R15]	"Digital Signatures using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)" ANSI X9.31-1998, American Bankers Association
[R16]	"FIPS PUB 46-3, Data Encryption Standard" October 25, 1999 (ANSI X3.92), National Institute of Standards and Technology
[R17]	"FIPS PUB 81, DES Modes of Operation" April 17, 1995, National Institute of Standards and Technology
[R18]	"FIPS PUB 180-3, Secure Hash Standard" October 2008 , National Institute of Standards and Technology
[R19]	"FIPS PUB 186-3" June 2009, Digital Signature Standard (DSS)
[R20]	"Public Key Cryptography using RSA for the financial services industry" ISO/IEC 9796-1, annex A, section A.4 and A.5, and annex C (1995)
[R21]	“Information technology – Security techniques: Data integrity mechanism using a cryptographic check function employing a block cipher algorithm” ISO/IEC 9797-1 (1999) , International Organization for Standardization
[R22]	“FIPS PUB 140-2, Security requirements for cryptographic modules” Mars 2002 , National Institute of Standards and Technology
[R23]	PKCS#1 The public Key Cryptography standards RSA Data Security Inc. 1993
[R24]	Security IC Platform Protection Profile, Version 1.0, reference BSI-PP-0035 (15.06.2007).
[R25]	Security Target Lite, Public - ST23YL80C of ST Microelectronics ANSSI-CC-2009/37
[R26]	Security Target Lite, Public - ST23YR80B/48B of ST Microelectronics ANSSI-CC-2010/01
[R27]	IEEE Std 1363a-2004 Standard Specification of Public-Key Cryptography
[R28]	FIPS PUB 197, The Advanced Encryption Standard (AES) U.S. DoC/NIST, November 26, 2001.
[R29]	CERTIFICATION OF APPLICATIONS ON “OPEN AND ISOLATING PLATFORM Paris, the 27th July 2012. Reference: ANSSI-CCNOTE/10EN.02deW10
[R30]	The NIST SP 800-90 Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revise) – March 2007

Ref	Document details
[R31]	Référentiel general de sécurité version 1.0 du 06/05/12
[R32]	Applications on ID ONE COSMO V7.1-S FQR 110 6268
[R33]	Note d'application 6 Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI)
[R34]	ANSI x9.62-2005 Public Key Cryptography for the Financial Services Industry The Elliptic Curve Digital Signature Algorithm (ECDSA)
[R35]	ANSI x9.63-2001 Public Key Cryptography for the Financial Services Industry Key Agreement and Key Transport Using Elliptic Curve Cryptography
[R36]	JIL-Guidance-for-smartcard-evaluation-v2-0
[R37]	ID-One Cosmo V7.1 Security Recommendations FQR 110 6029
[R38]	ID-One Cosmo V7.1 Reference Guide FQR 110 6028
[R39]	ID-One Cosmo V7.1 Pre-Perso Guide FQR 110 6027
[R40]	ID-One Cosmo V7.1 Application Loading Protection Guidance FQR 110 6267
[R41]	Note d'application 10 Agence Nationale de la Sécurité des Systèmes
[R42]	The Java Virtual Machine Specification. Lindholm, Yellin ISBN 0-201-43294-3
[R43]	Java Card 3 Platform Off-card Verification Tool Specification, Classic Edition Version 1.0. Published by Oracle
[R45]	Java Card System Protection Profile Collection Version 1.0b – August 2003
[R45]	Java Card System Standard 2.2 Configuration Protection Profile – PP/0305 Version 1.0b – August 2003