Security Architecture requirements (ADV_ARC)
for smart cards and similar devices
extended to Secure Sub-Systems in SoC
Appendix 1

This page is intentionally left blank

Table of contents

Contents

0 Preface

This document is the informative part of the Joint Interpretation Library document "Security Architecture requirements (ADV_ARC) for smart cards and similar devices extended to Secure Sub-Systems in SoC".

It contains examples for the type of information and level of detail to be provided in the ARC document applicable to this technical domain.

The preface of the document suggests additional terminology, abbreviations and examples of document structure useful for the ARC document. The main part of this document is at once:

- a template of the ADV_ARC document (printed in normal font)
- an explanation that can help in the understanding of the mandatory part (in box)
- a collection of examples (prefixed with the key word <**Example**> and ended by <>**)**

0.1 Glossary

This section suggests some terminology in order to help the developer describing the security architecture of the TOE. While the CC terminology (cf. [3] and [4]) focuses on description of security requirements these additional terms are intended for detailed description of the design and the implementation of the security architecture.

Security function – description *what* (in terms of action) the TSF, a subsystem, or a module does in order to meet one or more SFR (i.e. it is a subset of TOE security functionality).

Security property – invariant property of the TOE, the TSF, a subsystem or a module related to security. The generic security property of the TOE is non-bypassability of the TSF. The generic security properties of the TSF are domain separation, non-bypassability, secure initialization, self-protection as required by family ADV_ARC.

Security feature - combination of security functions implemented and security properties ensured on level of TOE, TSF, subsystem and module in order to prevent one or more attacks. Often the composition of a security feature only becomes clear when considering a specific attack path during vulnerability analysis.

Security mechanism - description *how* a security function (or its part) is implemented in order to meet an SFR or to enforce architectural soundness. The level of details is defined by purpose of modules (cf. component ADV_TDS.3 and higher).

Countermeasure – generic word, all means to protect the assets against the threats.

Architectural countermeasure – description *how* a security property of the TOE, the TSF, subsystems or modules is implemented or enforced by other means than functions or mechanisms. E.g. the TSF of a security integrated circuit implements light sensors (mechanism) to protect sensitive modules against light attacks. The physical layout (architectural countermeasure) – besides many other factors – exercises an influence on the effectiveness of this protection depending on but not changing the security mechanism of the light sensors.

Security service - a combination of security functions provided for the user. As an example the Security IC provides security services for the Security IC Embedded Software (e.g. cryptographic operations, random number generation). If the TOE is a composite TOE, the "platform" provides security services to the "application" (following the definitions of Supporting document CCDB 2007-09-001).

Note the term "security measure" means technical and organizational measures that ensure the security of the development environment and production environment; therefore security measures do pertain to the TOE but are not part of the TOE or subject of the ARC document.

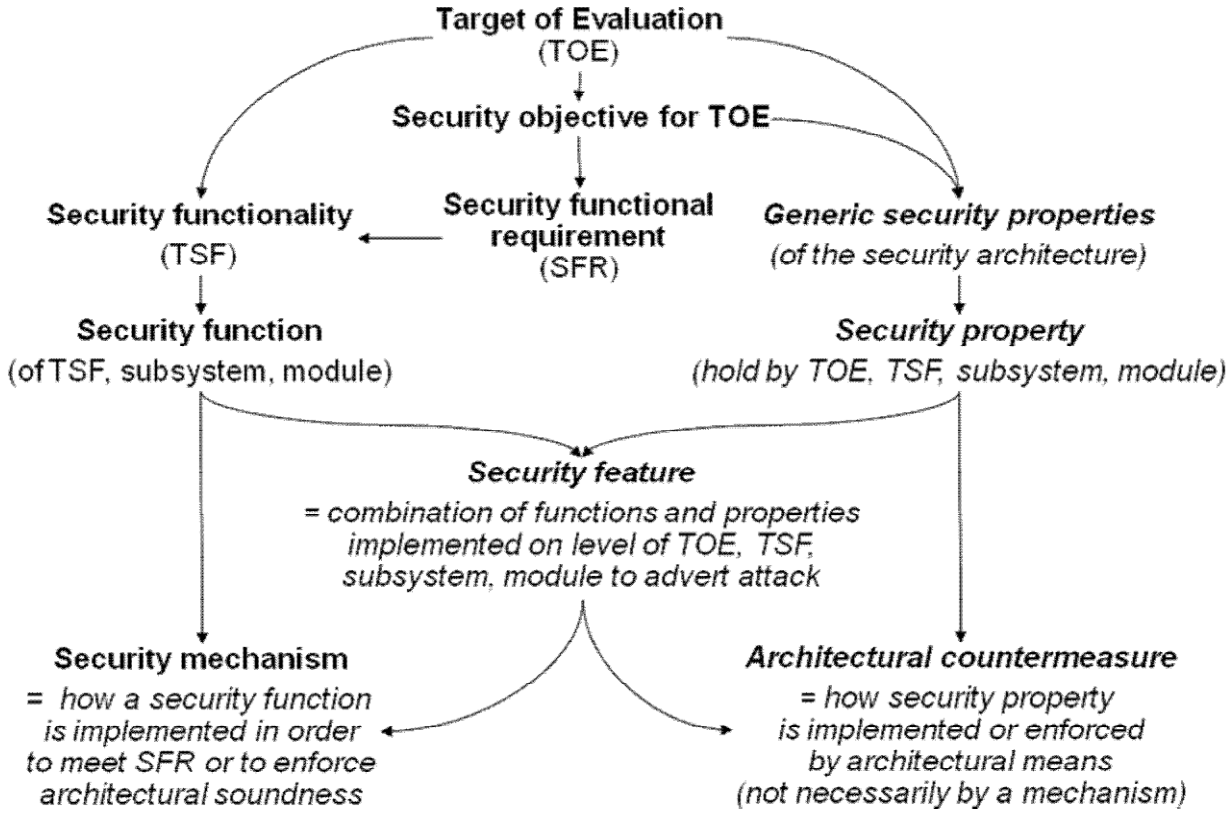


Figure 1 Suggested terms for security architecture description

The text printed in italic in figure 1 is not contained in CC / CEM but introduced here.

Note these terms will be used in the concrete context of the TOE, the TSF, the subsystems or modules. E. g. (cf. figure 2) a security service is provided to users (i.e. external entities) only while security functions may be provided for external users, for internal subsystems / modules or for both. A (complex) security function may comprise several (elementary) security functions e.g. a digital signature service includes RSA key generation and RSA signature generation. Even an elementary security function may be implemented by several security mechanisms, e.g. RSA key generation includes a key generation algorithm calculating corresponding private and public keys and uses the random number generation provided as service by the security IC. A security mechanism may be part of the implementation of one or more security functions or features, e.g. a random number generator provides random numbers for the security service random number generation and clock randomization as side channel countermeasure of the AES co-processor (cf. non-bypassability). A security mechanism may be used for enforcement of self-protection only like hardware memory encryption.

Note the CPU of the security IC is a SFR-supporting module of the security IC but implements together with the arithmetic co-processor the RSA algorithm in the SFR-enforcing module RSA signature. The arithmetic co-processor does not implement any security functionality on the hardware level.

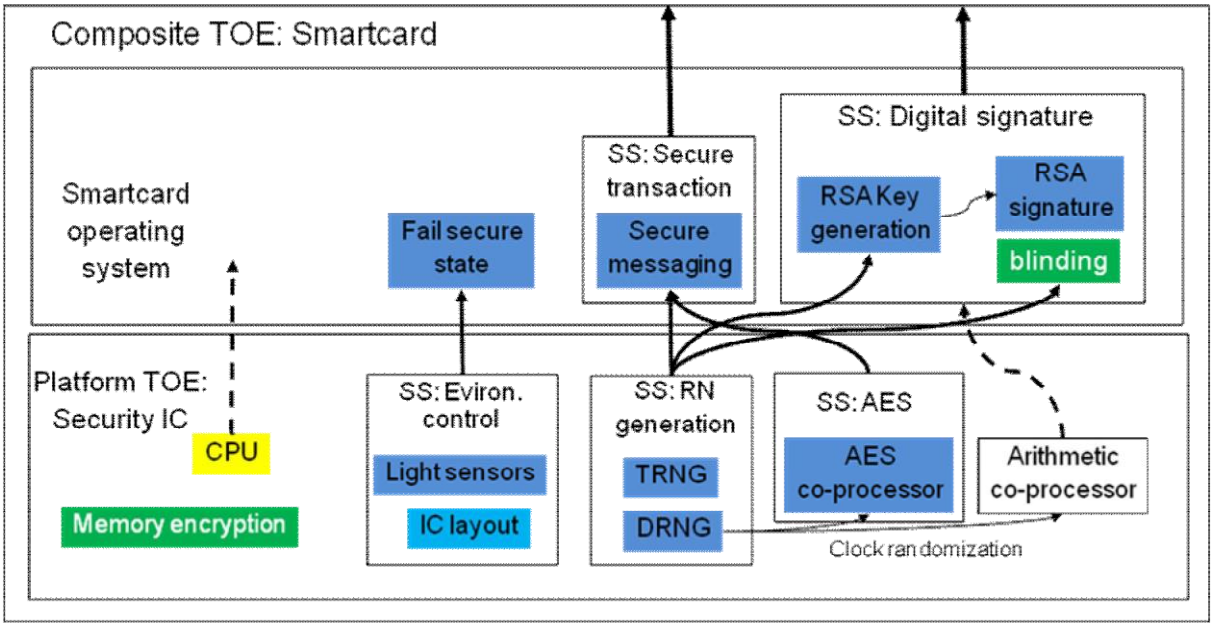


Figure2 Examples of security services, security functions, security mechanisms and architectural countermeasures

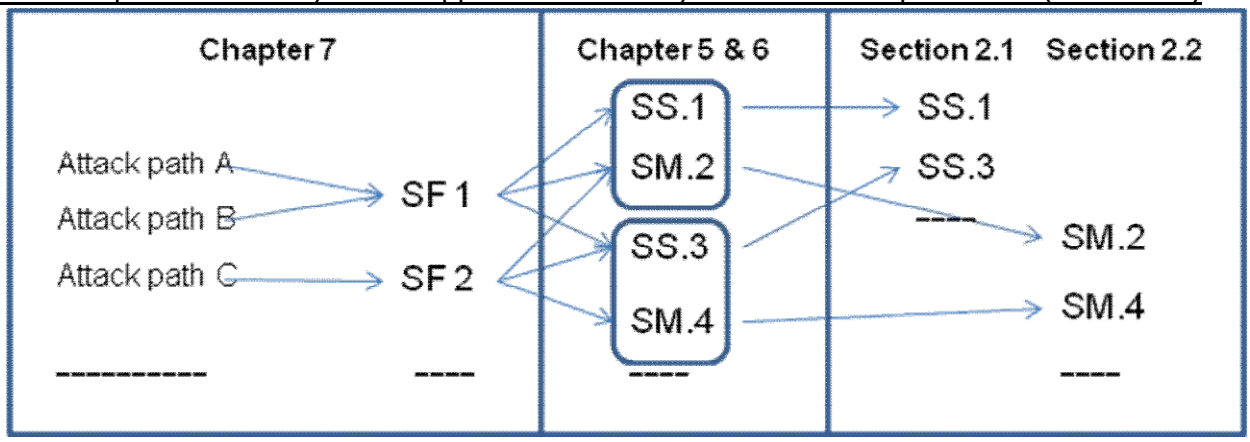
0.2 Abbreviations

SFR	Security Functional Requirement
SM	Security Mechanism
SF	Security Feature
SS	Security Service
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSS	TOE Summary Specification from ST
3S	Secure Sub-System
SoC	System on a Chip

0.3 Editing convention of the document

The structure of the current document reflects both a descriptive approach (security features, security functions / security functions, security mechanisms) and a demonstration of the architecture soundness checking the TSF behavior under attacks.

The following scheme illustrate this rational.



There are several possibilities to structure the ARC document. The suggested structure of the ARC document follows the generic security properties. The sequence of the chapters "Non-bypassability" and "Self-protection" is a matter of taste of the editor. They could be arranged also like this.

1. Security domains
2. Secure initialization
3. Non-bypassability
4. Self-protection

Here security features and security mechanisms enforcing architectural soundness may be described in the chapter 2 to 4 as appropriate. The demonstration of non-bypassability and self-protection may be more or less directly linked to the attack scenarios they are intended to withstand.

The current document starts with an overview of security functions, features and mechanisms. The chapter "Security domains" is based on the security target. If any security domains are identified their description should be used in the following parts of the ARC document. The enforcement of domain separation may be discussed in this chapter or later on depending on the security architecture of the TOE.

The chapter "Secure initialization" describes how the TSF is initialized. It describes the stepwise activation of TSF from the "down" state (e.g. power-off or after reset) into an initial secure state (i.e. when all parts of the TSF are operational) including possible temporary deactivation and activation of TSF. This chapter considers the description of security domain in the previous chapter. This chapter should describe and may include demonstration of specific aspects of non-bypassability and self-protection during secure initialization.

The chapter "Non-bypassability" demonstrates the security architecture of the TOE and the TSF from the external interfaces down to the subsystems, the TSF modules and their interaction. This provides a good overview on how the TSF works. But the ARC document could also discuss the countermeasure against specific bypass attack scenarios like side channel attacks.

The chapter "Self-protection" demonstrates the ability of the TSF to protect itself from manipulation from external entities that may result in changes to the TSF, so that it no longer fulfills the SFRs. It relates to the integrity and management of the mechanisms that constitute the TSF and to the integrity of TSF data. It discusses security mechanisms and their binding in order to prevent direct attacks. But the ARC document could also discuss the countermeasures against specific tamper attack scenarios like physical or logical manipulation.

The current document suggests a separate chapter discussing the intended TSF behavior under attacks. This chapter discusses single or the whole security architectural properties from the point of view of attack scenarios. This chapter is not intended as vulnerability analysis which focuses on the search of vulnerabilities, their exploitation and calculation of necessary attack potential.

Security Architecture (ADV_ARC) example

<Product Name Title>

Security Architecture description

1 Introduction

1.1 Purpose and Scope

This document contains the security architecture description for the <Product Name Title> as required by the CC part 3, chapter 12.1 as family ADV_ARC.

The TOE is made of: <TOE short description>
The Architecture description concerns: <the TSF>
The Security Services used are given in: <TSS of underlying platform>
The Security Services provided are given in: <TSS of the Security Target of this TOE>

The ARC document describes the security architecture of the TOE. Smart cards and similar devices are often evaluated in form of a composite evaluation [7]. A composite TOE is made of

- a platform layer constituted by the security hardware XX or by the embedded software YY running on the security hardware XX,
- an applicative layer constituted by the embedded software YY running on the security hardware XX or by application ZZ executed by the embedded software YY.

The ARC document of the security hardware describes the security architecture of a TOE providing unaided security functionality like physical protection, security services e.g. symmetric cryptographic operations, random number generation, and supporting functionality like arithmetic operations for asymmetric cryptographic operation. Similarly, the ARC document of a platform comprising security hardware and embedded software describes the TOE security architecture comprising of application independent security functions like access control and security properties like enforcing confidentiality of data which are more complex than those of the security hardware alone.

The security architecture of the composite product integrates the security services and security properties provided by the platform to the application and those implemented by the application itself. The security target of the platform describes the security functions and services provided for the application. The security target may describe the platform security architecture properties of self-protection and non-bypassability if it is compliant to ASE_TSS.2. The guidance of the platform describes how the application may use the security services of the platform in a secure way. But the ARC document of the platform is not necessarily available to the developer of the composite product. Therefore, guidance of the platform should support the application developer respective the composite product evaluation sponsor to develop and describe the security architecture of the composite TOE.

The ARC document makes reference to the TSF description given for ADV_FSP, ADV_TDS, ADV_IMP assurance classes. The FSP and TDS documentation focus on the complete and accurate instantiation of the SFR but may also describe the mechanisms implemented to enforce the security architecture properties. The IMP documentation makes the entire implementation representation of the TSF available and provides a mapping of (part or entire) TDS (and therefore of the instantiation of the SFR) to the implementation representation. The ARC documentation may but is not required to map the security architecture specific mechanisms (e.g. side channel countermeasures) to the implementation representation of the TSF.

Secure Sub-Systems (3S) in SoC – Integration

Though a 3S in SoC has many similarities in comparison to a SmartCard or similar devices, there are distinct differences that have to be considered in the ARC.

Very similar to SmartCards, a 3S in SoC, depending on the supported functional packages, provides unaided security functionality like physical protection, security services e.g. symmetric cryptographic operations, random number generation, and supporting functionality like arithmetic operations for asymmetric cryptographic operation. Similarly, the ARC document of a 3S comprising security hardware, the TOE security architecture comprising of host and composite software independent security functions like access control and security properties like enforcing confidentiality of data which are more complex than those of the 3S alone. The security target of the 3S gives a clear indication of the security functions and services provided to environment of the 3S.

In difference to a SmartCard, a 3S is typically integrated into a hosting SoC. The integration, design restrictions and the portability of evaluation results of an evaluated 3S to another hosting SoC have to be clear to integrating parties, evaluators and developers and is covered in the ETR for integration. The ADV_ARC will contain a rationale how the requirements as outlined in the Integration guidance will be followed up by SoC. If this is the initial first evaluation of the 3S in a SOC no ETR for Integration will be available.

A 3S may be utilized by a multitude of composite software, which may not be in control of one developer. Therefore, the composite software have to be isolated from each other, and also from the 3S. There is a mandatory vertical and horizontal separation of composite software, which has to be explained in the ARC documentation.

The ADV_IMP, ADV_FSP and ADV_TDS description shall have an equal level of detail as for SmartCards and similar devices.

Because of the distinct similarities, the additional requirements, explanations and refinements for a 3S in SoC in comparison to SmartCards and similar devices are defined as an addendum to the existing descriptions.

1.2 Documentation

For information and details beyond this document refer to the following specifications:

- Functional specification according to CC part 3, ADV_FSP.x ref xxx
- TOE design according to CC part 3, ADV_TDS.x ref xxx
- Implementation representation according to CC part 3, ADV_IMP.x ref xxxx
- Others …

2 Security services and Security mechanisms

Security functions describe what (in terms of action) the TSF, a subsystem, or a module does in order to meet one or more SFR (i.e. subset of TOE security functionality). The external visible security functions of the TSF are described in ADV_FSP. The security functions of subsystems and modules are described as behaviour and through interfaces.

Security mechanisms describe how a security function (or its part) is implemented in order to meet an SFR or to enforce architectural soundness. The security mechanisms implementing security functions are described as purpose of modules in ADV_TDS.3 and higher. The security mechanisms implemented in order to enforce architectural soundness are described in the ARC document or references are given to ADV_TDS document in the ARC document. Security mechanisms of this type may enforce domain separation and secure initialization, prevent non-bypassability and protect the TSF from tampering.

Binding of the security functions and all security mechanisms is very important aspect of the security architecture. E.g. a cryptographic key generation function may use a random number generator as mechanism, but may also need mechanisms for separate representation of generators for each security domain (cf. domain separation), generation of sufficient entropy of the internal states (cf. secure initialization), side channel resistance (cf. non-bypassability) and protection against tampering of the source of randomness or the internal state (cf. self-protection) well as means enforcing its usage for selected cryptographic functions (cf. non-bypassability).

The following chapter identifies example security functions and security mechanisms for the later description of domain separation and secure initialization and demonstration of non-bypassability and self-protection. Note the ARC document for a concrete TOE shall provide or reference to a description of the security functions and mechanisms more detailed than in the current template document.

2.1 Security Services provided by the underlying platform

In the case of a composite TOE the developer provides an overview of the TOE Summary Specification (TSS) defined in the Security Target of the underlying layer in this paragraph.

Smart security devices are generally based on an underlying certified platform (security IC or secure IC with embedded operating system) which provides its certified security services to enforce the global security of the product. Those essential services referring to security mechanisms of the underlying platform are recalled if necessary in this section as a short description focusing on properties enforced. The purpose is to ensure consistency of the overall document. It is therefore not recommended to list all underlying security services but only those that are used by the top layer.

Those mechanisms are already certified and generally described in the related IC datasheet or any other document provided to the TOE developer.

Note the application may also use and rely on correctness of functions of underlying platform which are not provided as security services and are not in the scope of the evaluation (e. g. arithmetic operations of the CPU used for cryptographic calculations).

The underlying platform provides the following Security Services (SS):

<Example> Security IC as underlying platform

SS.IC.Hardware cryptographic computation

Security IC performs cryptographic operations according to FCS_COP.1/XXX (e.g. two-key Triple-DES or AES encryption and decryption) ensuring confidentiality of the used cryptographic key and operated data (see IC datasheet chapter x).

SS.IC.Random number generation

Security IC provides unbiased and independent random values generated by a physical random number generator for use in cryptographic key generation, algorithms and protocols (see IC datasheet chapter x). The security IC is delivered with a RNG self-test software library detecting non-tolerable statistical defects of the generated random numbers due to aging of the entropy source.

SS.IC.Management of physical memory

Security IC provides Memory Management Service controlling access to memory areas and ensuring data storage isolation (see IC datasheet chapter x). Code running in System operating mode has full access to all EEPROM and RAM memory areas, special function registers and the management function itself. Code executed in Normal mode has access to defined EEPROM and RAM memories only.

SS.IC.Exception handling

Security IC provides exception handling initiated by and depending on by dedicated reason. The function XXX initializes and handles the exception.

SS.IC.Control of operating conditions

Security IC provides stable execution of the code and provision of the security services in the limit of defined operating conditions. Outside the defined operating conditions the security IC enters a fail secure state (internal reset). The operating conditions are described in the IC datasheet chapter x.

<this is only a partial list with some examples of SSs>

<Example> Security IC with IC Dedicated Software

SS.IC.Generic cryptographic library

The underlying platform comprise a security IC and a cryptographic library providing certified RSA and AES algorithms (see platform datasheet chapter x). The TOE under evaluation uses the AES library part only (see TDS documentation x chapter y).

<this is only a partial list with one example of SS>

<Example> Secure software underlaying platform

SS.SWPL.Secure object

The underlying platform provides containers dedicated to store sensitive data. These containers ensure protection of integrity of the stored data (see OS datasheet chapter x).

SS.SWPL.Secure erasure

The underlying platform provides a secure erasure of sensitive data which ensures that all related memory cells are overwritten, protecting against data leakage (see OS datasheet chapter x).

<this is only a partial list with some examples of SSs>

2.2 Security mechanisms of the TSF

This chapter includes a list of the security mechanisms of the TSF and a description or a reference to the description in the specific ADV_TDS document. How they are linked together (and with the Security Services of the underlying Platform) to prevent attacks and protect the TSF is described in chapters 5 and 7.

For the convenience of the reader descriptions that have been included here (with the references to other ADV documents added when possible) where found appropriate.

This section describes the security mechanisms provided by the TSF. In case of composite TOE these descriptions can make reference to the Security Services provided by the underlying platform to explain their behaviour.

Some aspects of the Security mechanisms are already described in ADV documents:

- FSP provides the TSFI description. The TSFI may or may not describe the external behavior of a security mechanisms A link to the associated description is in this case very relevant to avoid rewriting a full description.

  Ex: PIN management, secure protocol, access control…

- TDS provides a more detailed description of the design of the product. Some security mechanisms dedicated to ensure correct and secure execution could be described here. A link to the associated description is in this case very relevant to avoid rewriting a full description.

  Ex: a function providing a security check, …

The security mechanisms enforcing security properties of the TSF (like domain separation, secure initialization and non-bypassability) and self-protection but not directly implementing SFR may be described in the ARC documentation or in TDS and referenced in the ARC documentation.

Some mechanisms protecting the TOE are spread all over the code and possibly only at a code level. They must be described at a level consistent with the ADV_IMP, but only a generic description of the principle is provided here.

The other parts of the ARC document shall describe how and where they are used in order to allow for analysis of completeness. Reference would be made to these descriptions in the following chapters of this document.

<Example> Security Mechanisms of the Security IC

Special modules SM.IC.TDES co-processor

Security IC implements a hardware cryptographic co-processor performing two-key Triple-DES encryption and decryption. The co-processor implements non-bypassability countermeasures against timing attacks (i.e. the time of execution is independent on the key, the data and encryption/decryption operation), power analysis and emanation analysis.

SM.IC.AES co-processor

Security IC implements a hardware cryptographic co-processor performing AES encryption and decryption with key length 128 bits, and 256 bits. The co-processor implements non-bypassability countermeasures against timing attacks (i.e. the time of execution is independent on the key, the data and encryption/decryption operation), power analysis and emanation analysis.

SM.IC.Physical RNG

Security IC implements a physical random number generator compliant to AIS31 (see IC datasheet chapter x) generating unbiased and independent random numbers. The RNG implements selfprotection mechanisms detecting failure and non-tolerable statistical defects of the entropy source. The RNG self-test software library is integrated in the TOE under evaluation (cf. TDS document x chapter xx and IMP document y part yy). This software is not side-channel resistant and therefore the tested random numbers must not be used for other purposes than testing.

SM.IC.Memory Management Unit

Security IC implements a Memory Management Unit (see IC datasheet chapter x). Access attempts outside the defined memory areas or to special function registers cause exception x (see IC datasheet chapter y).

Countermeasures enforcing Self-Protection- Sensors

The device has several types of exception sensors monitoring its extrinsic operating parameters such as voltage, frequency, temperature, and light, which are totally independent from each other and from the other functionality of the TOE. The exception sensors guarantee the device to be operated under the specified conditions. In case specified operating conditions are not fulfilled the sensors generate an internal device reset.

The sensors can be enabled or disabled in order to allow testing of the device functions beyond the sensor threshold. By testing devices in production outside the sensor limit the correct function of the device and its sensors at the sensor limits is guaranteed. After delivery the hardware does not allow the embedded software to interfere with the sensors.**SM.IC.Low and High Frequency Sensors**

The Smart Card Controller cannot be operated at low clock frequency ($f_{CLK} < f_{CLK(LFS)}$, LFS=Low Frequency Sensor). At low frequencies a permanent RESET is performed. If the clock frequency raises above the high frequency limit ($f_{CLK} > f_{CLK(HFS)}$, HFS=High Frequency Sensor), reset will be executed as well.

SM.IC.Voltage Sensors

The Smart Card Controller includes a power-on-reset circuitry controlling the dedicated power-on and power-off sequence.

When the voltage $V_{DD} < V_{DD(LVS)}$ (L_{VS}=Low VDD! Sensor) or $V_{DD} > V_{DD(HVS)}$ (H_{VS}=High V_{DDs} Sensor) a reset will be executed.

SM.IC.Temperature Error Sensor

The device supplies a temperature error sensor monitoring the operating temperature. If the temperature drops below $T<T_{(LTS)}$ (LTS=Low Temperature Sensor) or if the temperature raises above $T>T_{(HTS)}$ (HTS=High Temperature Sensor) an internal reset will be executed.

SM.IC.Light Sensor

There are a number of light sensors located on the chip that are designed in a way that assures normal device functionality under specified conditions, but invokes a sensor reset, if the device is exposed to strong light emitters.

Countermeasures enforcing Self-Protection - Detectors SM.IC.Active Security Routing

(Description of the security routing)

SM.IC.Parity checks

The security IC implements parity checks of data

(1) stored and read from EEPROM and RAM

(2) transferred on bus between ROM and CPU, EEPROM and CPU, RAM and CPU, <…> as self-protection mechanisms.

SM.IC.Reset

The security IC implements internal reset mechanism setting of the program counter to 0.

Countermeasures enforcing non_bypassability SM.IC.Desynchro HW

The device features a desynchronization mechanism, by adding fake clock cycles and jitter. This feature is enabled by setting the register XXX to following value: XXX. Such countermeasures aim at disturbing an accurate synchronization; it is activated when <to be specified><reference to AGD_OPE>.

SM.IC.Filter

A filter is applied to the current flow in such a way that the power consumption of the chip is smoothed during execution of critical operation.

Further Countermeasures

For functionality, which contribute to the security of the TOE but is not described as a Security Mechanism in the ADV documentation, a detailed description of the functionality of the security mechanism can be given in the following. (The following description serves as an example.)

SM.IC.Arithmetic co-processor

The security IC implements an arithmetic co-processor for calculation of sum, product and square of two operands modulo a number (see IC datasheet chapter x). The execution time of the arithmetic operation does not depend on the operands and the modulus. The TOE under evaluation implements <all, partly> the countermeasures described in the security IC security guidance x chapter y, cf. TDS documentation chapter z.

<this is only a partial list with some examples of SMs>

<Example> Security mechanisms of secure software platform

Global security

SM.SWPL.Redundancy

It consists of executing the same operation twice to check that both of them give a coherent result. The second operation can also be different, by being for instance an inverse computation.

SM.SWPL.Shadow Memory for NVM write parameter

The security mechanism stores the address and length information for the NVM write parameter. Detailed description of the implementation is given in x.

SM.SWPL.Time-constant execution

Loops and other control structures are designed in such a way that execution time is independent of the associated control value if this information is considered as sensitive. For instance a loop computation on an array is performed from the first to the last element even if the expected result is known in the meantime.

SM.SWPL.Desynchro SW

To disturb any attacks requiring an accurate synchronisation, some fake code is executed at random. This functionality is ensured by the function XXX and activated by calling the function with the parameters xxx

SM.SWPL.Masking

To hide sensitive values during their manipulation, masking data techniques prevent against any estimation of the concerned values. It consists of applying a mask to a value, performing computation with this masked value and inferring the result for the sensitive – unmasked – value. In this case, no computation is performed on the sensitive value and no leakage via side channel can be exploited

Security Services provided by the TOE

These mechanisms are building blocks used to build security functionality provided by the TOE. They are generally internal to TSF modules and therefore often described in the TDS. Nevertheless if the TOE is a platform for composite certification their external behaviour is more probably described in the FSP since these mechanisms will be provided to the final developer.

SM.SWPL.State machine

The sequences of commands are checked to avoid bypassing of steps. If, at the beginning of each command processing, a command sequence isn't correct then the state machine returns an error. The current state is protected using SS.**IC.Control of operating conditions** (see ADV_TDS §x.x).

SM.SWPL.Transaction

Atomic transaction ensures the integrity of data stored in persistent memory even in case of power loss or perturbation. When opening a transaction context (see Transaction_open in ADV_TDS §x.x) an unused persistent memory area is allocated for the data to be stored. The transaction writes this data in the reserved memory area (see Transaction_write in ADV_TDS §x.x). If the transaction updates a data object then the original data is kept valid in the persistent memory until the write operation is finished, and the old data is erased and the persistent memory area is deallocated after the write operation. The transaction is finalized by building a valid data object containing the successfully written data (see Transaction_finalise in ADV_TDS §x.x).

SM.SWPL.File integrity

Files created are composed of a data section and a control section. This control is a MAC computed over the data of the file using SS.IC.Hardware cryptographic computation. This MAC is synchronized with the data section using SM.SWPL.Masking (see ADV_TDS §x.x and ADV_FSP
§x.x).

SM.SWPL.Secure erasure

The underlying platform provides a secure erasure of sensitive data which ensures that all related memory cells are overwritten, protecting against data leakage (see OS datasheet chapter x).

Functional security

These mechanisms implementing security functionalities provided by the TOE actually allow describing secure initialisation and discussing non-bypassing, self-protection or domain separation but are not provided to the final developer in case the TOE is a platform. Their external behaviour will probably be described in the FSP.

SM.SWPL.Authentication tries mechanism protection

Usage of the TOE is allowed after a successful authentication. The number of authentication attempts is limited to a maximum value. This mechanism is designed to ensure that the value cannot be modified or disclosed by an attacker by using e.g SS.IC.Exception handling , SS.IC.Control of operating conditions, SM.SWPL.Transaction (see ADV_FSP §x.x).

SM.SWPL.Access control protection

This mechanism is designed to detect abnormal operations that might indicate an attack by an external entity by checking integrity of data accessed (using e.g. SM.SWPL.File integrity, SS.IC.Management of physical memory SS.IC.Exception handling,) and verifying access rights (see ADV_FSP §x.x).

SM.SWPL.Secure loading

Loading of keys is based on a secure channel using 3DES (SS.IC.Hardware cryptographic computation) and AES (SS.IC+CL.Generic cryptographic library)and session keys based on a random value (SS.IC.Random number generation) When loaded on the TOE, keys are stored in a secure container (SS.IC.Management of physical memory) (see ADV_FSP §x.x).

SM.SWPL.Unblocking

TOE is initially blocked and need a first authentication (based on SS.IC+CL.Generic cryptographic library) to unblock it and authorize TOE usage (see ADV_FSP §x.x).

<this is only a partial list with some examples of SMs>

3 Security domain separation

In this chapter it is explained what the different kinds of domains supported by the TSF are, how they are defined (i.e. what resources are allocated to each domain), how no resources are left unprotected, and how the domains are kept separated so that active entities in one domain cannot tamper with resources in another domain.

Security domains for Security ICs

For Security ICs it can be assumed that security domains exist such as Test Mode and Operational Mode (or similar). In Security Targets compliant to BSI-PP-0084- 2014, these are defined through FMT_LIM.2. Implementation of these security domains is considered in other ADV families and is not repeated here.

Other security domains might exist, for example due to an implemented Memory Management Unit. In this case they should be described in the Security Target by Security Functional Requirements (SFRs). If a security domain is not explicitly described within the Security Target, it has to be described here.

<Example> Security domains for Security ICs – Test domain and Operational domain

The SFR FMT_LIM.2 defined in BSI-PP-0084-2014 separates the Test domain and the Operational domain.

- In the Test domain the dedicated software of the TOE is available e.g. for tests of the TSF by the manufacturer. The Embedded software is not available because it is intended for the operational use.
- In the Operational domain the embedded software is available for execution by the user but dedicated software is not available. That is because the embedded software may violate the security policy of the user e.g. by reading, calculating and reporting checksums over the physical memory areas including secrets.

Note if the dedicated software of the TOE is available but with limited capability this will be dealt under non-bypassability (cf. FMT_LIM.1 in BSI-PP-0084-2014).

<Example> Security domains for ICs – System mode and User mode

The memory management unit (**SM.IC.Memory Management Unit**) allows enforcement of

- System mode of program execution with access to all memory areas and security functional registers including the management registers of the MMU,
- User mode of program execution with limited access to memory areas and no access to security functional registers including the management registers of the MMU,
- Transition between these modes by exception handling.

Note that the Security IC maintains these two security domain while the operating system running on this platform may support further refined security domains within the user mode.

Security domains for composite smart cards

Untrusted entities and resources are defined by the limits and content of the TOE. For example:

In the case of the TOE being an "integrated product", where the Security IC Embedded Software consists of native code that implements both OS and application behaviour without demarcation between them, there are no domains because all actions are brokered by the TOE. The application is performed by the TSF that maintains only data structure to keep user's data separated. (cf. basic configuration in [11])

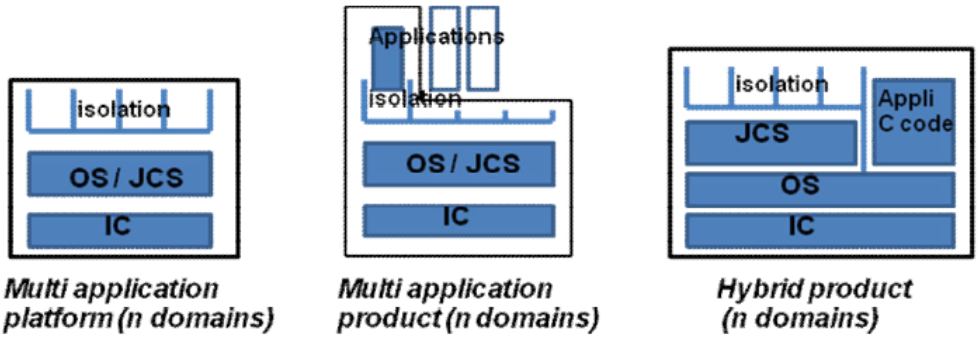
In the case of the TOE being a "layered product" where the Security IC Embedded Software consists of an "OS Layer", potentially with integrated application behaviour, and an "Application Layer" on top of it, there are two domains. The OS provides a separation mechanism between itself and the Application Layer as well as services to the Application Layer. If the TOE does not contain application code in the "Application Layer" the domain separation exists as a service-offer by the platform to a composite product built on it. (cf. extended configuration in [11])

In the case of the TOE being a multi-applicative platform, applications are untrusted entities potentially active. An example is the JavaCard Platform described in the JavaCard System PP. The TOE is responsible for card resource management and applet execution. Applet isolation is achieved through the applet firewall mechanism that confines an applet to its own designated memory area. Thus each applet is prevented from accessing fields and operations of objects owned by other applets or data owned by the TOE itself.

In the case where the TOE includes application(s) resident on such a multi-applicative platform the evaluated and non-evaluated applications of the product are maintained separately thanks to the service offered by the platform. The capability of post issuance downloading ("open platform") does not introduce any differences in terms of domain separation.

Other structures can be found that mix the previous cases, for example a product that is a hybrid with respect to a JavaCard platform and an isolated applicative C-code.

The following figures illustrate these various cases.



Multi application platform (n domains) *Multi application product (n domains)* *Hybrid product (n domains)*

For cases where the TOE depends on the IC or a platform to play a role in domain separation, that sharing of roles must be made clear by referencing the IC or platform security services.
For the composite smart card product, there are also different life-cycle modes as initialization, personalization or application mode. If these modes involve interfaces and code execution that are part of the TOE they can support different sets of domains with different mechanisms for domain separation.
When the domains and domain separation are described in the Security Target by Security Functional Requirements they will not be described in further details as implementation of these security domains is considered in other ADV families.
If the security domains are not explicitly described within the Security Target, they have to be described here.
This description shall take into account all SFRs claimed by the TOE. That is to say the access to the resources allocated to a domain must be covered by SFRs and SFRs corresponding to a function of resource control must correspond to a domain separation mechanism.

The document "Application of Attack Potential to Smart Cards" lists attacks specifically relevant for domain separation:
- Ill-formed Java Card Applets
- Violation of firewall between applets

<Example> Security domains for a Java Card smart card - Applets

Each applet or applet package loaded on the TOE can be considered as residing and performing in a security domain; separation between them is controlled through the firewall. The JavaCard VM, associated execution piles running on behalf of this applet, the parts of memory that contain the byte-codes, and the objects belonging to the applet are the resources allocated to the security
domain of this applet.

Indeed, the TSF with the firewall controls information flow at runtime. It controls object sharing between different applet instances, and between applet instances and the Java Card RE.

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: the applet instances context and the Java Card RE context, which has special privileges for accessing objects. No distinction is made between instances of applets defined in the same package: all of them belong to the same active context and therefore to the same security domain. In contrast, instances of applets defined in different packages belong to different contexts and therefore to different security domains. Each object belongs to the context (defined as a security domain) 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. In contrast, 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 Y of document X. Those rules enable controlled sharing of objects through interface methods, that the object's owner explicitly exports to other applet instances. Put differently the object's owner explicitly accepts to share these objects upon request of other applet instances invoking the interface method.

Security domains for 3S in SoC

In difference to a traditional security IC, several Composite Software can run on the 3S at the same time. There may be e.g. several OS / Composite Software that need separation. The Composite Software is not able to protect itself from other Composite Software, therefore the platform has to provide the necessary means to enforce Security Domain Separation.

There has to be some control mechanism provided by the TSF, as described in [13] by the Domain Separation. Composite Software shall not be allowed to compromise the TSF, even if only one is present.

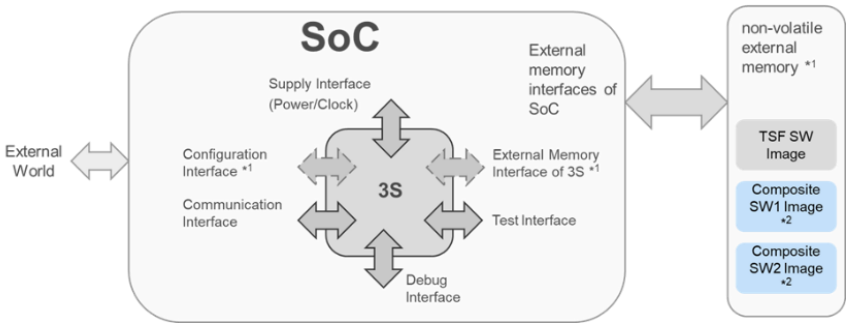


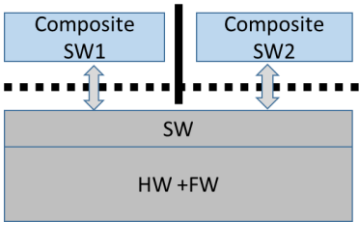
Figure 3 A SoC (light grey) with 3S (dark grey) and different composite software (light blue) in external memory.

In this chapter, it is required to provide a description on how TSF defines security domains and manages them in order:

- to perform access control to avoid any unauthorized access to TSF resources (code and data) by any other entity running on TSF.

- to perform access control to composite SW to avoid any unauthorized access to composite SW (code and data) from TSF or any entity running on TSF.

Figure 4: Principle of vertical and horizontal separation provided by the 3S (grey) of composite software (light blue)



This covers two topics: Self-protection (protecting the 3S against a maliciously acting Composite Software) and security domain separation (protecting Composite Software running in parallel from each other).

<Example> Security domains for 3S – Composite Software

Each composite software loaded on the 3S can be considered as residing and performing in a security domain; separation between them is controlled through a sandbox, establishing isolation of the composite software from the 3S (horizontal) and other composite software (vertical).

Furthermore, the 3S ensures that any memory containing TSF code is not shared between composite software.

During the execution of a composite software the 3S tracks the activity of the active composite software that is currently performing an action. This information allows the 3S to govern the activity and to ensure that no composite software acts without notice of the 3S.

The 3S provides dedicated interfaces to the composite software to be able to offer data and/or code to other composite software. The other composite software has to actively choose to use this data and/or code.

The 3S acts as an arbiter, e.g. each peripheral has an assigned active owner which is tracked by the 3S and access has to be explicitly granted by the 3S. A dedicated secure release procedure ensures that all assets that might be used in conjunction with a peripheral (keys for a cryptographic co-processor, buffers containing intermediate results etc.) are erased securely before any other owner may interact with the peripheral.

<Example> Security domains for 3S – External Memory

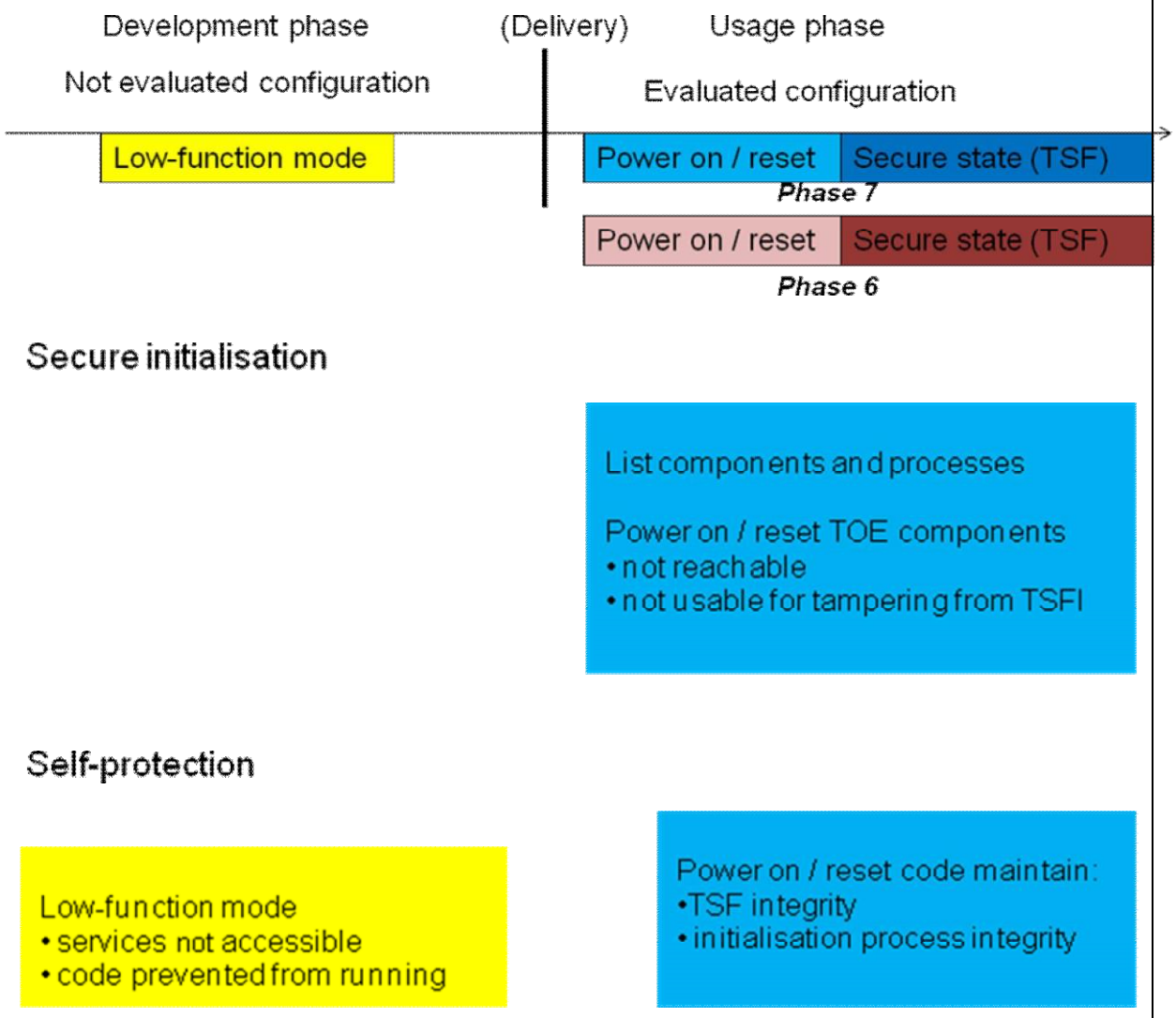
The 3S utilises external memory in addition to the internal memory, which is not under direct control of the 3S. The 3S therefore is responsible for the protection of the communication with the external memory, validation of content and protection against tampering with externally stored data/code.

All relevant content of the external memory is stored in encrypted form (AES256). Each image is encrypted with its own unique key. The 3S is responsible for the secure handling of the keys.

To ensure that the correct and up-to-date images are loaded, hashes (SHA3) of the respective image, which are stored in the internal memory of the 3S, are compared with the hashes of the images which are loaded. Execution of an image is only possible after successful validation of the respective image. The hashes of the images are updated before and after each authorised modification the NVM and therefore ensuring that no old data will be loaded. Furthermore, this enables the 3S to ensure that the content of the NVM was only altered by authorised modifications. Any detection of altered hashed will lead to the conclusion, that the image or NVM are compromised.

To avoid replay attacks the 3S implements a counter with 4 tries, that is raised on each failed attempt to validate the hash of a known image. If the counter reaches its limit the image is considered as compromised.

4 Initialization / start-up



The developer describes the start-up sequence with reference to the code modules that are executed to perform the initialization function. It points out the functions that are not part of the TSF and then not described yet in ADV_TDS.
The description focuses on the overall strategy, how the TSF is initialized in a secure way and all security features initialized during the start-up (e.g. backup recovery, sensitive areas integrity check, fault counters …).
For the functions that are not described in ADV_TDS, it is explained why they are not reachable when the TSF is in the secure operational state or if they are reachable why their external interfaces cannot be used to tamper with the TSF.

The description of how the initialization process guarantees the TSF integrity at the end and how it is itself protected against modification is part of the chapter “Self-Protection”. However for clarity it could be described here and referenced in this chapter.

<Example> Secure ICs – Secure initialization and environmental control

As long as the operating conditions of the device in terms of voltage, frequency and temperature do not reach the specified range the whole IC is kept in reset. After reset the device starts with the internally functionality test as …..

(It has to be described why the boot sequence (start up sequence) is implemented such that a secure state will be reached.)

Excerpt from BSI-PP0084-2014
The Security Architecture description of the TSF initialization process shall include the procedures to establish full functionality after power-up, state transitions from the secure state as required by FPT_FLS.1 and any state transitions of power save modes if provided by the TOE.

<Example> Start-up test of RNG

The RNG (**SM.IC.Physical RNG)** performs the following self-tests during start-up before delivering any random number:

(1) Breakdown test of the entropy source,

(2) Statistical test of the internal random numbers

If the start-up test fails the TSF enters a fail secure state.

The following text shall provide more detailed TOE-specific description.

<> <Example> Composite smart card

The initialization/start-up process switches the TOE from the power-off (down) state into an initial secure state. This process is invoked in two cases:

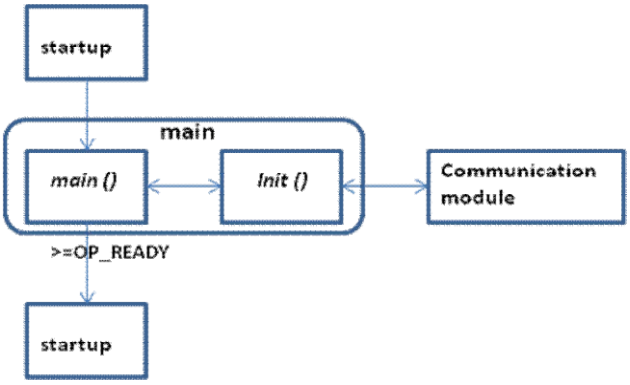
- Cold reset: the power is applied to the TOE
- Warm reset: the RESET signal is sent to the running TOE

The start-up process includes the IC initialization, the OS & JCS initialization and the selection of the default application in java state.

(Cold or Warm) reset is the only external interface of the TOE initialization function. In other words, there is no other way to invoke this process.

Vcc (for power-on) and RST (for RESET) contact points are the only external interfaces of the TSF initialization. There is no other way to execute this process.

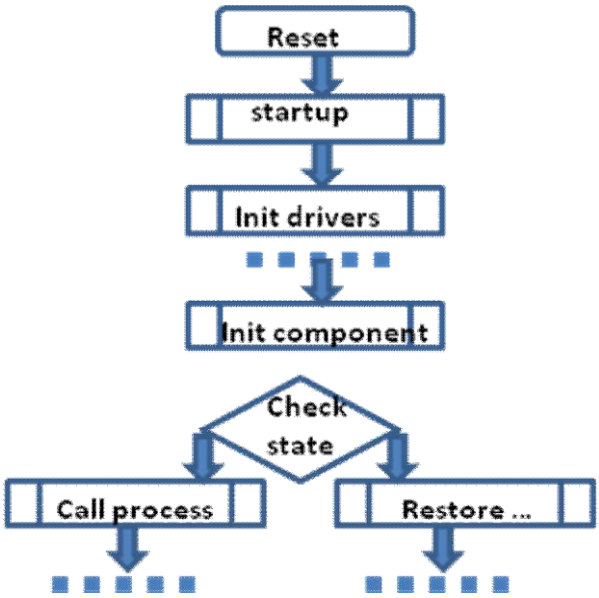
The following diagram shows an overview of the reset sequence (component dependencies):



1. **startup**: the startup component is responsible for the initialization of the chip. It initializes the CPU registers, the security features and it clears the RAM.
2. **main:** the main component is in charge of initializing the OS&JCS. It includes the drivers' initialization, the initialization of the memory manager module, the communication module, and the cryptography module. Once the OS & JCS initialization is achieved, the main module calls the JCRE component.
3. **jcre**: the JCRE component receives and dispatches APDUs (the platform state being OP_READY or later). At the first jcre invocation, the default applet is selected.

The **main** module always receives control after startup execution in order to initialize the native operating system and the Java Card operation system.

Following diagram shows the initialization flow chart:



The start-up is composed of a boot sequence (including self-tests) and different security initializations. It ensures that the chip is operating and protected against perturbation before sending the ATR.
The software component *boot.c* allows checking and initializing the security services of the component:

- SS.IC.Hardware cryptographic computation (see also SM.IC.TDES co-processor, SM.IC.AES co-processor)
- SS.IC.Random number generation (see also SM.IC.Physical RNG) It is protected by the following Security mechanism:
- SM.SWPL.State machine
- SM.SWPL.File integrity

The other HW sensors are activated automatically at power up of the chip and before the *boot.c* execution.
The start-up executes also the *ErrorRecovery.c* file handling the Transaction mechanism. At the end the SM.SWPL.Transaction is initialized.

The Init component allows the checking (including self-tests) and initialization of the Security services. The following Security services and mechanisms are used for protection against side channel by-pass and perturbation:

- SM.IC.Filter

- SM.SWPL.Masking
- SM.SWPL.File integrity

The TOE reaches a secure state after a successful completion of the initialization sequence, because all security services mechanisms, security services and software modules are initialised in proper order.
Provide an explanation of the fact that the secure initialization of the composite is consistent with the guidance and used services of the platform.

As long as the following security mechanism SM.7 is not set up, the TOE protects the start-up sequence against fault injections only by the sensors (cf. SM.IC.5 to SM.IC.8) and filters (cf. SM.IC.12 Filter) of the chip.
...
The start-up sequence enforces that the SM.6 transaction mechanism is always invoked and performs a proper roll-forward or roll-backward before any software modules which depend on the transaction mechanism are invoked.

The *ErrorRcovery.c* file cannot be executed in the secure state because it can be called only by the startup component.

<>

<Example> 3S – Recovery when TSF code is stored in external NVM during start-up sequence

The 3S utilises external NVM to store TSF code within a TSF SW image. The image is loaded during the start-up sequence to execute.

The image is stored encrypted and hashes of the image are stored internally. The hashes are compared with the image, which is to be loaded. The data, its integrity and freshness are checked before loading the TSF SW image.

In case of loss of integrity, authenticity or freshness is discovered during start up sequence, at least a part of TSF code cannot be loaded and executed. The TSF is executed in “degraded mode”. The mode of the 3S than indicates that a failed start-up sequence was performed.

After successful authentication of the administrator for recovery it is possible to operate in a maintenance mode to recover full TSF functionality. There are several possible operations from refresh operations on the NVM or providing a valid back-up of the TSF SW image.

As a last line of defence, the 3S stores a minimal fall-back image internally which guarantees the successful operation of the core TSF functionality. This minimal SW image has not the same functionality as the externally stored TSF SW image, but provides all necessary means to protect against manipulation, disclosure of code and data.

5 Self-protection

The document "Application of Attack Potential to Smart Cards" lists attacks specifically relevant for self-protection

- Physical Attacks
- Overcoming sensors and filters
- Perturbation Attacks
- Attacks on RNG
- Buffer overflow or stack overflow (depending on attack scenarios)

5.1 Self-protection and initialisation process

The description of how the initialization process guarantees the TSF integrity at the end and how it is itself protected against modification could be provided in the Chapter "Initialization / start up " for clarity . In this case reference is made to this Chapter.

5.2 Self-protection and low function mode

When the TOE is initialized in a low function mode and then transitions into the evaluated configuration the developer has to explain how the TSF is protected against this initial code. This is typically what could happen in the TD SC&SD during the life cycle of the TOE when a specific code or a partial implementation is running during the development phase to build the final configuration that will be delivered

<Example> Composite smart card

A specific code as a light OS or a loader is used to load the remaining part of the TOE at development site.

<>

<Example> 3S – Degraded mode

The 3S Firmware provides the necessary TSF to import a TSF software image. If there are any issues detected with the import of the TSF software image, the 3S can enter degraded mode.

The 3S provide recovery operations in the degraded mode, like deleting broken images, download and export of a new image, to recover full functionality. Only after successful authentication it is possible to perform recovery operations.

Export of software may use different keys, so only this chip can read the image. There shall be no unencrypted export of data.

5.3 Self-protection in full operational state of the TSF

<Example> Security IC - Physical tampering

The self-protection against physical tampering comprises:

(1) Passive security features as combination of features of the used IC technology and specific implementation of the TSF increasing the necessary effort of physical manipulation.

(2) Active security mechanisms detecting manipulation of the TSF, e.g. SM.IC.various sensors, SM.IC.Active Security Routing and SM.IC.Parity checks; and reacting on detected manipulation with reset (by SM.IC.Reset).

<*provide further TOE specific description here*>

<Example> Security IC - Environmental controls

<*provide further TOE specific description here*>

<Example> Security IC – Perturbation

<*provide further TOE specific description here*>

<Example> Security IC – RNG self-protection

The RNG implements self-protection mechanisms detecting failure and non-tolerable statistical defects of the entropy source.

<*provide further TOE specific description here*>

<Example> Composite smart card

The electrical profile of the operation is varying from one execution to the other by using the desynchronisation mechanisms SM.IC.Desynchro HW & SM.SWPL.DesynchroSW. This effect makes it difficult to reproduce an attack at the same step of the computation.

Redundancy (SM.SWPL.Redundancy) applied to critical operation preserves the result and prevents falsification.

Detection of error by redundancy or detection of intensive light by the chip (SM.IC.Light sensor) triggers an exception. An adaptive reaction depending on the sensitivity of the code is managed by the exception handling mechanism (SS.IC.Exception handling) ensuring the preservation of a secure state.

Detection of file corruption by SM.SWPL.File integrity when the file contains sensitive data triggers an erasure of the file thanks to SM.SWPL.Secure erasure.

<>

<Example> Isolation between 3S and hosting SoC – External Memory

The 3S utilises external memory in addition to the internal memory, which is not under direct control of the 3S. The 3S therefore is responsible for the protection of the

communication with the external memory, validation of content and protection against tampering with externally stored data/code.

…

The external memory is only accessible to the 3S and unshared with the hosting SoC. The SoC has no connection to this external memory. Due to the fact that the 3S is the only entity interacting with the memory, the SoC has no way to alter content. Only external attackers have to be considered in this scenario.

<Example> 3S – Isolation between 3S and hosting SoC – Operational mode

The 3S has dedicated interface for configuration of the TSF, which are disabled after locking the test mode.

Entering the test mode requires mutual authentication, with a try counter of 4.

Entering the test mode clears all relevant memories and caches, ensuring that the test mode cannot be misused. Furthermore, the functionality in test mode is restricted such that it is not possible to characterize the TSF.

The test mode is locked by blocking write access to distinct configuration packages utilizing the 3S integrated MMU.

After the test mode is locked, it is not possible to alter the life-cycle of the 3S.

<Example> 3S – Power Management

The 3S in SoC shares its power source with the hosting SoC. To allow a secure operation of the 3S a voltage stabilisation within the 3S is implemented. If the input voltage is detected as outside of the specification, a security reset is performed.

To protect the 3S from leakage attacks via the shared power source, the power consumption is masked by using dual-rail pre-charge logic. If necessary additionally algorithmic masking of the power consumption is performed.

<Example> 3S – Recovery when TSF code is stored in external NVM during execution

Can be different or equal to the situation during start-up, has to be described by the developer how it was solved.

…

The 3S utilizes external NVM to store TSF code within a TSF SW code. It has to be checked at least before execution if the code was successfully loaded.

The code is stored encrypted and hashes of the code are stored internally. The hashes are compared with the code, which is to be loaded. The code, its integrity and freshness are checked before loading the TSF SW code.

In case of loss of integrity, authenticity or freshness is discovered, at least a part of TSF code cannot be loaded and/or executed. The TSF is executed in “degraded mode”.

After successful authentication of the administrator for recovery it is possible to operate in a maintenance mode to recover full TSF functionality. There are several possible operations from refresh operations on the NVM or provision of a valid back-up.

As a last line of defence, the 3S stores a minimal fall-back image internally which guarantees the successful operation of the core TSF functionality. This minimal SW image has not the same functionality as the externally stored TSF SW image, but provides all necessary means to protect against manipulation, disclosure of code and data.

<Example> 3S – Usage of (passive/secure) ext. NVM shared with SoC

The 3S has two sets of NVM. An intern secure memory, which is only accessible to the 3S and an external unsecure memory, which is shared with the hosting SoC.

The memory controller for the shared memory is part of the 3S and allows to allocate a dedicated address range for the 3S. The address range XXXXXXX to YYYYYYY is solely reserved for the 3S, the SoC shall not have access to those data segments.

All relevant content of the external memory is stored in encrypted form (AES256). Each image is encrypted with its own unique key. The 3S is responsible for the secure handling of the keys.

…

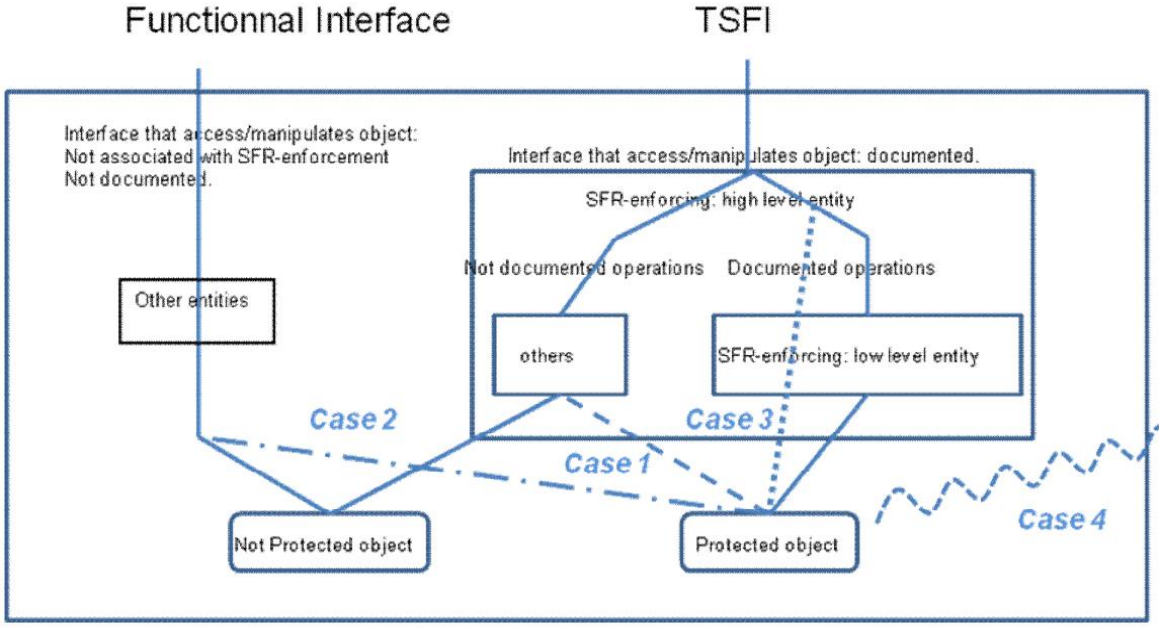
6 Non-bypassability

All modes or operations of TSFI are documented in ADV_FSP and all interactions between modules are documented in ADV_TDS. No further description is required.

When Functional Interfaces (i.e. external interfaces that are not TSFI) exist the developer shall list them and explain either why they have no interaction with the TSF or why they are not a path for violation of security objectives.

A demonstration that the TSF prevents bypass of the SFR enforcing functionalities is given by providing a description on how the TOE reacts in the presence of the relevant attacks listed in the document "Application of Attack Potential to Smart Cards" and bringing a conclusion. This demonstration is provided in the Chapter "TOE protection in presence of attacks".

Protection against exploitation of an insufficient design or implementation by an attacker having logical access to the TOE and protection against confidentiality objective violation by side channel analysis attacks are part of this Chapter.



Case 1: exploit undocumented mode or operation of TSFI
Case 2: exploit undocumented functional interface
Case 3: exploit insufficient design or implementation
Case 4: exploit side channel

The document "Application of Attack Potential to Smart Cards" lists attacks specifically for non-bypassability

- SPA/DPA – Non-invasive retrieving of secret data
- Higher Order DPA
- EMA Attacks
- Exploitation of Test features
- Bypass authentication or access control
- Buffer overflow or stack overflow (depending on attack scenarios)
- Ill-formed Java Card applications
- Information gathering
- Software attacks

- Command editing

6.1 TSF always invoked

<Example> Physical protection of the Security IC

The physical protection surface builds a continuous perimeter but with different security mechanisms appropriate for the relevant physical attacks

(1) the front side of the Security IC is protected by sensors SM.IC.Light Sensor, SM.IC.Active Security Routing, and SM.IC.Parity checks,

(2) the back side of the Security IC is protected by sensors SM.IC.Light Sensor, and SM.IC.Parity checks and reacting on detected manipulation with reset (by SM.IC.Reset). in combination with internal reset (by SM.IC.Reset) when physical tampering is detected.

The sensors and the active security routing are located where critical modules are implemented (see TDS documentation x chapter y and IMP documentation z). The sensors <list of sensors> are always active and will not be disabled even in power save mode.

The parity checks controlling the modules <list of modules> and cannot be disabled.

<Example> Environmental control of the Security IC

The sensors SM.IC.various sensors control the environmental operating conditions continuously causing an internal reset (cf. SM.IC.Reset**)** when violation are detected. The correct operation within the controlled environmental condition is demonstrated by characterization tests (cf. ATE documentation x). The sensors and the interaction with internal reset cannot be disabled.

<Example> For smart cards with JavaCard System

Due to the EAL4+, all the operations and modes of the TSFI are documented in ADV_FSP and ADV_TDS. So the exploitation of undocumented mode or operation of TSFI for bypassing an SFR_enforcing entity is not possible.

Section XXX has identified the security domains (SD) of the Javacard platform and shown that all these SDs enforce their own isolation. In particular it is not possible for the SD involving functional interfaces belonging to an applet that is not included in the TOE to access any protected object managed by a security domain involving a TSFI; the exploitation of an undocumented functional interface is therefore not possible.

<>

6.2 Side Channel

The security mechanisms and how they are working together to avoid information leakage or how they distort the information in such a way it is not exploitable are described below:

<Example> Illicit information flow due to internal data transfer

FDP_ITT.1 Basic internal transfer protection and FPT_ITT.1 Basic internal TSF data transfer protection in BSI-PP-0084-2014 [9] require the TSF to protect user data and TSF data from disclosure when it is transmitted between separate parts of the Security IC (i.e. different memories, the CPU and other functional units (e.g. a cryptographic co-processor) are seen as separated parts of the TOE). In this case side channel protection is directly addressed by SFR. In other protection profiles side channel resistance is required by an extended component SFR

(e. g. BSI-CC-PP0059-2009-MA-02 [10] defines FPT_EMS.1). The developer may decide to deal with all aspects of side channel resistance in the ARC document.

<Example> Cryptographic co-processors of the Security IC

SM.IC.TDES co-processor and SM.IC.AES co-processor performing unaided cryptographic operation implement security mechanisms preventing side channel attacks.

Note, arithmetic co-processors do not implement cryptographic algorithms and cannot be claimed as SFR compliant to CC part 2 in security targets[1]. Therefore they are not part of the TSF and addressed in the security architecture and vulnerability analysis of the Security IC.

<Example> Composite smart card

SM.IC.TDES co-processor, SM.IC.AES co-processor, SM.SWPL.Time-constant execution and <list of security mechanisms in cryptographic implementations> prevent illicit information flow on sensitive data due to the execution timing of a critical operation.

When SS.IC.Random number generation is active random numbers are constantly generated. This mechanism adds a perpetual noise that is added to the current consumed by the chip. Moreover SM.IC.Desynchro HW creates disturbance in the normal current profile of operations and SM.IC.Filter is blurring the final power consumption.

SM.SWPL.Secure loading protects the keys when they are loaded to be used by the crypto library.

The mechanisms SM.IC.Desynchro HW, SM.Desynchro SW and SM.Masking hide the sensitive data with a statistical noise when the attacker tries to get information by correlation between power consumption and manipulated data.

<>

<Example> Side channel protection in 3S and hosting SoC

Even if the interfaces of the 3S or the surface of the 3S are not accessible, measurements of the hosting SoC have to be considered.

Two aspects have to be considered for the integration of a 3S:

1. The first integration of the 3S in a hosting SoC,
2. The re-use of the 3S in another hosting SoC.

Both, the integration guidance and the ETR for Integration have to be considered, if present.

Especially the design specification and integration guidance states that the 3S has to be embedded such that a set of assumptions / requirements have to be considered:

- placement within the hosting SoC,
- layers above the 3S,
- Termination of interfaces,
- …
- *<list of requirements and assumptions of the design specification and integration guidance>*

For each requirements:

[1] The developer might define an extended component in order to describe the functionality of the arithmetic co-processor.

- *<Rational how those requirements and assumptions are fulfilled in the given SoC.>*

Changes of interfaces of the 3S in SoC have to be considered for the integration into another SoC. The 3S as of now has the following interface defined:

- *<list of interfaces>*

For the instantiation into the current SoC no changes to the interfaces were performed.

7 TOE protection in presence of attacks

The analysis assesses the effectiveness of the TOE Security Features and/or the Security Mechanisms (SM) of the TOE to resist against different attack methods. The CC supporting document "Application of Attack Potential to Smart Cards" [8] serves as a reference to ensure covering state of the art attacks. In the following, examples are presented along the lines of the current [JHAS] document at the time that this Guidance document was written.

<Example> Secure IC

Physical Attacks

The attack is directed against the IC and often independent of the embedded software (i.e. it could be applied to any embedded software and is independent of software counter measures). The main impacts are:

- Access to secret data such as cryptographic keys (by extracting internal signals)
- Disconnecting IC security features to make another attack easier (DPA, perturbation)
- Forcing internal signals
- Even unknown signals could be used to perform some attacks

The potential use of these techniques is manifold and has to be carefully considered in the context of each evaluation.
(Continue description, name/describe all the Security Mechanisms and/or Security Features involved and how they are supportive to counter this attack.)

<Example> Composite product – Self-protection Perturbation Attacks

The attack will typically aim to make cryptographic operations weaker by creating faults that can be used to recover keys or plaintext, or to avoid or change the results of checks such as authentication or lifecycle state checks or else change the program flow.

The typical external effects on an IC running a software application and the reaction of the TOE are as follows:

Modifying a value read from memory during the read operation: the value held in memory is not modified, but the value that arrives at the destination (e.g. CPU or coprocessor) is modified. This may concern data or address information

- The sensitive data are stored in files. The fault generated in these files by the perturbation is detected by SM.SWPL.File integrity and the content of the file is erased by SM.SWPL.Secure erasure to be no more used.
- When the perturbation modifies the address where the data is read in such a way an access is provided to a different Security Domain, this violation is detected by SS.IC.Management of physical memorythat trigger an exception handled by SS.IC.Exception handling that leads on a reset.
- When the perturbation generates an abnormal operation on data the SM.Redundancy will detect the fault.
- Modifying the program flow: the program flow is modified and various effects can be observed such as skipping an instruction, inverting a test, generating a jump, generating calculation errors.

- Modifications of the program flow in the critical section of the code. Irrespective of the various effects, the attacks are always covered by one of the following mechanisms working together: SM.Redundancy, SM.Transactions, SM.Access Control.

<>

<Example> Composite product – Non-Bypassability – Side channel

The TOE exhibits power consumption, which is a function of the commands executed and the data used, as both the shunt current in the switching process and the capacitive recharging is dependent on the process taking place at that instant. Various methods of inferring the data being processed from analysis of the power consumption are known (e.g. SPA or DPA).

For this purpose it is first necessary to resolve the power consumption (also by averaging of repeated measurements) to the point that the information becomes visible. This attack consists of the two steps measurement and analysis.

The Security Mechanisms SM.n , SM.n+1, SM.n+m , … ensure that the data transferred via the bus is encrypted and prevents direct correlation of stored data and power consumption.

<>

<Example> Composite product – Non-bypassability- Insufficient design Information gathering and protocol attacks

This type of attack tries to use the protocols in ways that were not intended by the protocol developer or to send commands that the smartcard does not expect in its current state.

The API is characterised by a set of n commands, each having a method signature characterised by the parameters given in chapter x of the document y. The range of values for the parameters is given in the implementation of each command. A centralised command processor verifies the command parameter value and checks it against allowed values. If the delivered parameter is out of range, the command response will be the error code xy....” Module XX in the TDS is devoted to the handling of these aspects

<>

8 Bibliography

[3] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model; CCMB-2017-04-001, Version 3.1, Revision 5, April 2017

[4] Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Components; CCMB-2017-04-002, Version 3.1, Revision 5, April 2017

[5] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements; CCMB-2017-04-003, Version 3.1, Revision 5, April 2017

[6] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology; CCMB-2017-04-004, Version 3.1, Revision 5, April 2017

[7] Supporting Document Mandatory Technical Document Composite product evaluation for Smart Cards and similar devices, May 2018, Version 1.5.1

[8] Application of Attack Potential to Smartcards, June 2020, Version 3.1

[9] BSI-CC-PP-0084-2014 Security IC Platform Protection Profile with Augmentation Packages Version 1.0

[10] BSI-CC-PP-0059-2009-MA-02 Protection Profile for Secure Signature Creation Device - Part 2: Device with Key Generation, June. 2016

[11] Protection Profile Embedded Software for Smart Secure Devices Basic and Extended Configurations (reference ANSSI-CC-PP-ES for SSD, version 1.0, 27 November 2009)

[12] BSI-CC-PP-0099-V2-2020 Java Card System - Open Configuration Protection Profile Version 3.1, April 2020

[13] BSI-CC-PP-0117-TBD Sub-System in System-on-Chip (3S in SoC) Protection Profile

<developer documents>