IBM Enterprise PKCS#11

Firmware identifiers `dada00eb` and `e41c1444`

Security Target

EAL 4

IBM Research — Zürich and IBM Böblingen/Poughkeepsie

2017-08-30

Document revision: 273, last archived: 2017-08-30 14:59:45 (UTC)

Contents

1 Introduction

This document is a security target that defines the assets, threats, security assumptions, security functional requirements, security assurance requirements and rationale for the IBM Enterprise PKCS#11 firmware resident within IBM hardware security modules (HSMs) of model 4767 and 4765. This is version 1 of the security target, last updated 2017-08-30 14:59:45 (UTC), file revision 273. The security target corresponds to the shortened EP11 version identified as `dada00eb` (4767) and `e41c1444` (4765).

See section 8.1 for the description of full module configurations.

1.1 Security Target introduction

The IBM Enterprise PKCS#11 firmware (EP11) is an implementation of the industry-standard PKCS#11 cryptographic service provider API version v2.20 [PKC04] and some algorithmic extensions [PKC09, PKC14], adapted to requirements typical in enterprise servers. The EP11 firmware provides a stateless backend, relying mainly on host-resident, encrypted datastores to maintain sensitive state [VDO14], while presenting services as a regular HSM-based PKCS#11 implementation. When loaded as an application within suitable IBM HSMs, EP11 firmware provides a set of cryptographic services functionally identical to those specified by PKCS#11 with minimal host-library assistance.

Extended capabilities provided by hosting HSMs are either transparent to PKCS#11 applications, or are presented through implementation-specific additions through standard PKCS#11 extension interfaces. In addition to services derived from PKCS#11, the TOE provides an additional set of administrative services for management of keys, attributes, usage restrictions, and infrastructure — at a level more privileged than Security Officers of PKCS#11.

The TOE scope excludes host code, including drivers translating between PKCS#11 and EP11 interfaces. These transformations simply move traffic between serialization formats, and do not provide cryptographic or security-relevant functionality.

1.2 TOE type

The TOE represents the full firmware stack as loaded into an applicable HSM, *inheriting any physical protection as an environmental feature* afforded by code instantiated within HSM boundaries. Note: the TOE implements functions for physical protection which have not been subject to this evaluation.

The functionality includes cryptographic engines — see section 1.4.5 — which differ between 4767 and 4765, but export identical cryptographic functions to the host.

In this instantiation, the TOE is categorized as a "Cryptographic Module" assuming dedicated physical protection from the hosting infrastructure — implying operations in an access-controlled, physically protected environment.

The TOE includes services like the update of software that are restricted to administrative users. The assumption is made that those services are not used by such a user.

While we describe multiple hardware configurations, the PKCS#11-level EP11 firmware functionality is identical. Differences in the underlying infrastructure are known only to some engine-aware code, the only entity aware of details of internal functions.

1.3 TOE usage and major security features

The TOE, in the context of this security target, provides the following security features:

- Generation and distribution of cryptographic keys, including controlled import and export capabilities.

  Note that most uses of "export" in EP11 are not for transport to other cards. Objects or state stored on the host is expected to be retrieved by another EP11 module, possibly the originating one. This export from an HSM does not constitute "export" in the PKCS#11 sense — i.e., `WrapKey()` encrypting a key with another user-controlled one.

- A compliant implementation of most PKCS#11 security functions — "functional calls" — with minor adaptations unique to the hardware/software components of the TOE (section 8.2).

  Note that a fully transparent implementation of PKCS#11 also requires modest additional work transforming calls on the host, which is not further discussed in the context of the TOE. These additional steps simply de/serialize data without performing cryptographic operations.

- Secure storage and management of cryptographic keys during their HSM-internal lifecycle.

  While key-lifecycle management includes tracking and logical state of each key, it specifically lacks expiration based on time. While TOE environment contains an internal real-time clock, time-based expiration is not currently tracked. The essentially-stateless operational model of enterprise keystores implies that per-module key expiration is of limited value (expiration SHOULD happen consistently at the host level, not within multiple, uncoordinated HSMs).

- Secure management of key attributes, including association with keys, and restrictions on usage of keys.
- Creation and verification of encrypted and authenticated — "wrapped" — enclosures for sensitive data ("blobs") to be stored on untrusted hosts
- Creation, management, and lifecycle control of "wrapping keys," (WKs) top-level symmetric keys encrypting host-resident sensitive data. Maintenance of authentication — MAC — keys corresponding to each active WK.
- Random-number generation, supplying an internally generated bitstream intended for cryptographic purposes.
- Internal maintenance and separation of multiple "domains" (partitions)
- Internal maintenance and separation of user "sessions"
- Management of external administrator identities — i.e., *public keys* — of privileged operators authorized to modify security attributes beyond those accessible to PKCS#11 security officers.
- Secure management of administrative and usage-control attributes.
- Communication of audit-relevant events to host, as input to system-wide audit facilities. HSM-resident maintenance of certain audit-relevant features, for lower-assurance audit events where complete loss through tamper-induced destruction — which is an asynchronous, uncontrolled event — of such events is acceptable.
- Consistency-checking and other error checking of underlying crypto service provider functionality, including implicit coverage of any cryptographic hardware engines, through a dedicated set of tests

The TOE is implicitly influenced by any key-destruction initiated by the underlying tamper-detection mechanisms in hardware, both terminal ("hard tamper") and recoverable ("soft tamper") events. Depending on event type, the TOE may or may not be an active participant in event reactions. Specifically, tamper-induced zeroization of sensitive data within IBM HSMs is performed entirely by hardware, without firmware assistance or even cooperation.

1.4 TOE description

1.4.1 Physical scope of TOE

The TOE is the full firmware stack of the HSM plus algorithms implemented in an ASIC and FPGA that are used within the TOE. Physical instantiation, tamper response and other infrastructure circuitry are functions of the HSM enclosure (the latter as part of the TOE environment).

The externally observable interface of the physical instantiation is the external industry-standard PCIe bus. Note that in the claimed operational environment of the TOE the PCIe bus interface is not directly accessible by anybody. Access to the TOE is restricted to access by the zSeries functions.

TOE services are available as a typical request-response control/dataflow through the hosting HSM, providing responses to host-initiated requests. In addition to matched pairs of requests and responses, the backing HSM may generate traffic asynchronously, such as providing diagnostics.

The TOE is identified by the respective firmware identifier for the 4765 or 4767 model as provided in section 8.1 "Specification of full TOE configuration" of this document.

1.4.2 Logical scope of TOE

The logical scope includes TOE firmware, including the EP11 application, cryptographic service provider and HSM-internal configuration data. These components are stored in HSM-internal storage, or within ASIC/FPGA in their entirety, in a combination of persistent code and data storage. During servicing requests, the TOE may access request-integrated copies of key material inserted by the host, copies of which are then under control of the TOE.

Code within the appropriate IBM HSMs (4767 and 4765) is field-upgradeable, without cooperation of any previously loaded applications, by the underlying system firmware — "Segment 1", specifically "Miniboot", in IBM HSM terminology. The

operating system or applications, including all components of EP11, are incapable of updating code. Firmware updates need to be digitally signed; the TOE verifies the correctness of signatures before updates. Update functionality is not within the scope of this evaluation — except for verifying that an invalid signature on downloaded code updates is detected by the TOE and the installation of the software update is rejected.

The HSM-instantiated form of EP11 consists of the following groups of functionality:

1. Core EP11 functionality, implementing our documented PKCS#11-like services: the equivalents of PKCS#11 commands, queries, or other administrative functionality.
2. Cryptographic provider, providing cryptographic services, and abstracting away the specifics of HSM engine access.

   Note that the cryptographic provider of Enterprise PKCS#11 is an HSM-specific instance of CryptoLite for C (Clic), the official OS crypto service provider of most IBM Systems Group (i.e., server) operating systems.

   In IBM HSMs, the Clic instance is optimized for PowerPC processors, and utilizes hardware-based acceleration for applicable HSM engines, with an external API identical to that of software-based versions.
3. *Cryptographic engines provided by the hosting HSM,* used by the cryptographic provider (this is opaque to the EP11 core)
4. *Random-number generation,* supplying an entirely internally generated, hardware-originating stream, conditioned, and post-processed within the enclosure.

   Note that the physical noise source used to seed the random number generator is not part of the TOE but part of the TOE environment.
5. Infrastructure interaction, including reaction to HSM-provided events, request/response channel management, and other external entities unique to hosting HSMs
6. Dispatcher and thread management, the top-level entity reacting to host-initiated traffic (requests) and submitting responses through infrastructure-provided channels
7. A number of processing threads, each servicing requests assigned by the top-level dispatcher.

   Request processing includes both services derived from PKCS#11 functional calls, administrative queries or commands.
8. Audit infrastructure, constructing an unmalleable, hash-chain-based sequence of audit events, logging security-relevant EP11 events.
9. Other infrastructure software, such as self-test routines and firmware assisting startup (Fig. 1)
   (a) Power-on selftest (POST) code integrity-checking infrastructure, in multiple levels, gradually increasing coverage
   (b) Bootstrap control (Miniboot0). This code is ROMmed, not modifiable.
   (c) Security-state management (Miniboot1), including integrity-checking loaded firmware. Other functionality, as described detail below, is dormant in the evaluated configuration.

   See section 1.4.4 for more detail of this segmentation.

The list of features specifically excludes firmware-update capability during TOE operations. For practical purposes, IBM HSMs are non-modifiable runtimes during EP11 operations — even if firmware may be updated, but only as an offline operation. *In the configuration described by this Security Target, we assume that HSM infrastructure services are unchanged, and no code update happens during the operation of EP11 firmware.* (As a consequence of this limitation, we prohibit Miniboot services from executing any *commands* changing Miniboot-state. Note that such firmware updates would happen outside the scope of EP11 itself; we in fact impose this restriction on HSM infrastructure.)

Access to TOE services is provided over the PCI Express (PCIe) bus, submitted as selfcontained requests serializing request parameters, prefixed by transport headers. The interface of the request/response flow is fully documented through an implementation-neutral, standardized interface, in a dedicated "wire specification", which completely specifies formats and de/serialization rules for all traffic observable at the PCIe bus.

The TOE treats all host-originated data, including full serialized structures and its embedded components, as untrusted. *Denial-of-service attacks on the host or its I/O infrastructure, preventing access to the TOE or corrupting requests, are not considered: they prevent proper requests from reaching the TOE* but they MAY NOT be detected/protected against by the TOE.

The TOE provides the following main categories of security functions:

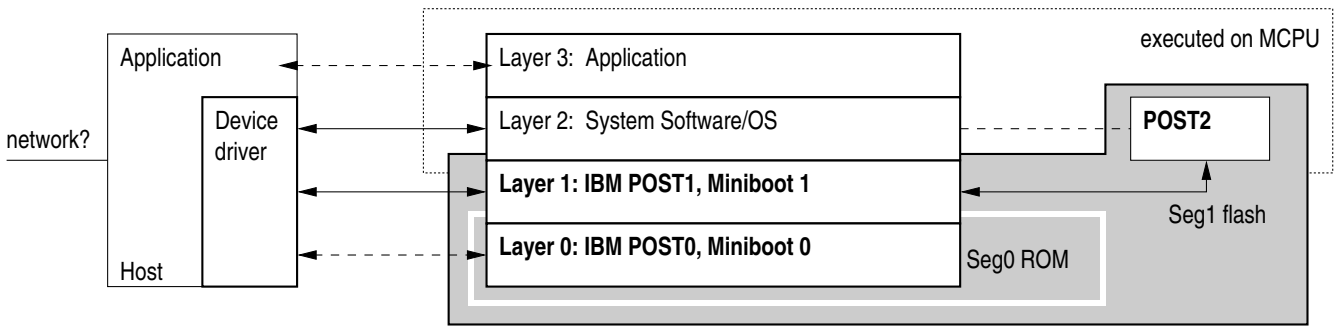

Figure 1: Module software architecture

Encryption, decryption, authentication, management of host-resident, sensitive, host-opaque objects as the step required to retrieve keys from requests

This category involves creation and incremental maintenance of any host-opaque, exported objects.

Generation, distribution of cryptographic keys including controlled import and export capabilities

Random-number generation supplying an entirely internally generated, hardware-originating stream, conditioned, and post-processed within the enclosure. The random-number generator may provide bytes to the host, or use them directly to generate cryptographic keys within the TOE.

Functional calls implementing the EP11 equivalents of PKCS#11 cryptographic services

Secure storage of HSM-resident keys if they are committed from host-resident to HSM-internal form ("semi-retained" keys). Once retained, these keys are referenced through handles and not available for subsequent export, but may be used by functional calls, or removed.

User authentication through session management, for functional calls, corresponding to PKCS#11 `Login` and `Logout` services in a way consistent with TOE assumptions

Administrator authentication through public-key signatures, for administrative services (no PKCS#11 equivalent)

Note that state-changing administrative services — administrative *commands* — are authenticated by external entities, and the TOE only *verifies* signatures on such authenticated data. Actual sensitive operations — signature generation — happen outside the TOE.

Administrative services, operations outside PKCS#11 scope. Generally, these services operate at higher privilege levels than any PKCS#11 operation, even if several of them interact with PKCS#11-visible objects.

Administrative services includes the following:

1. Creation and management of WKs
2. Transportation of card state
3. Management of administrative usage-control attributes, including restrictions based on key strength, key type, usage scenario, and other limitations on functionality. Restrictions are further subdivided into module- and domain-level attributes, with well-defined precedence rules for checking and enforcement.

Functional access control combining usage restrictions from generic administrative settings, "control points" of domains, and usage controls embedded into wrapped objects. These usage restrictions may interact with PKCS#11 capabilities, but have no equivalent PKCS#11 counterpart.

The TOE ensures separation between domains and between sessions. Separation between domains is achieved with domain-specific wrapping keys.

Administrative access control combining statically defined or attribute-dependent usage and access restrictions for HSM-resident state

Auditing supplying host-visible audit events, and short-term low-assurance events within the HSM

Self-tests verifying the integrity of HSM components, including the cryptographic provider and implicitly any hardware accelerators

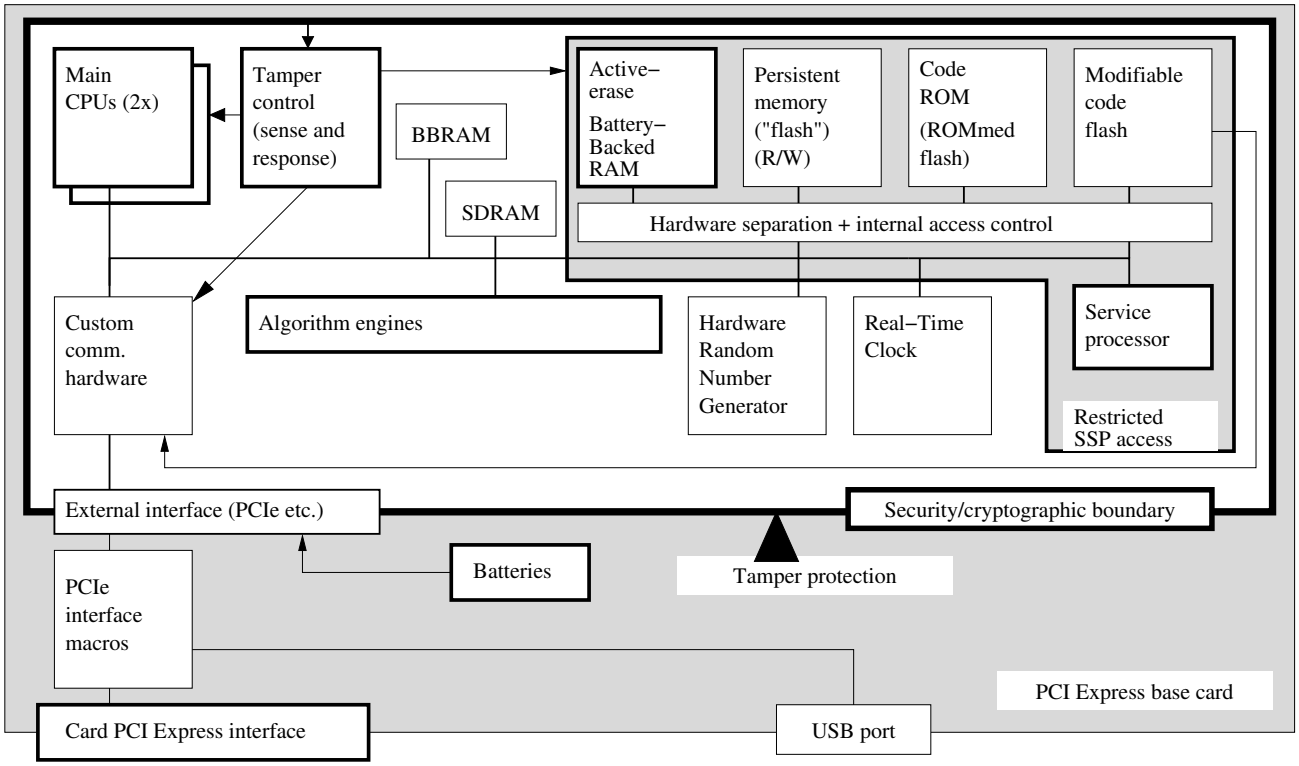

Figure 2: Module hardware architecture, with directly connected components on the hosting PCIe board (when deployed on an HSM)

1.4.3 Externally observable TOE functionality levels

TOE functionality forms clearly disjoint, hierarchical sets of services. In decreasing order of capabilities, the following categories of services are available:

1. Module-level administrative services, capable of managing properties of the entire module. In addition to maintenance of the most privileged, module-level settings, module administrators mainly manage the identities of lower-level — i.e., domain — administrators (section 7.1.7).

   Module administrators have no PKCS#11 equivalent, and are not expected to submit functional, non-administrative requests to the TOE. In a typical production setup, module administrators are expected to interact with the module relatively infrequently.

2. Domain-level administrative services, authorized to manage one particular domain, including PKCS#11-visible attributes within that domain. They are not authorized to influence module-level properties, and multiple sets of them coexist without the capability to influence other domains.

   Domain administrators are not expected to submit functional, non-administrative requests to the TOE.

3. Functional — non-administrative — PKCS#11 services, bound to sessions. These services are influenced by the administrative setup of the domain hosting them, but they may not change any administrative setting.

   Session management may change persistent TOE state — i.e., list of sessions — but not administrative structures.

4. Functional PKCS#11 services using keys, when used without sessions. Services available at this level match those of session-bound requests, but session-bound objects are rejected and may not be used. (In other words, session-less objects correspond to PKCS#11 "token objects" — those not bound to any specific PKCS#11 user.)

   Functional services in this category may change persistent TOE state — such as cached key entries, or list of semi-retained keys — but not administrative structures.

5. Functional services requiring no use of keys, such as hashing or random number generation. These services generally correspond to PKCS#11 ones, particularly those which may require PKCS#11 sessions, but no keys.

   Functional services in this category may change persistent TOE state, such as advancing the state of the random-number generator within the TOE-embedded crypto service provider Key-related or administrative structures are not accessed.

6. Queries, incapable of modifying TOE-visible objects.

State-changing administrative actions must be signed by administrator keys (table 5), therefore *administrator signing keys are implicitly trusted as authentication devices,* representing the administrator controlling the proper signature key.

The TOE does not distinguish between PKCS#11 security officers and users, as it lacks objects which would reflect that distinction. Any PKCS#11-level entity is less privileged than administrators.

Note that queries are generally available to any host entity capable of submitting requests to the module. The distinction between multiple host entities, and access control restricting use of queries, is out of TOE scope.

1.4.4 TOE software architecture

Internal software is divided into four layers. The base two layers, and a stub in the third layer control security and configuration of the module:

- Layer 0: Permanent POST 0 (Power-on Self Test) and Miniboot 0 (security bootstrap). This code is in ROMmed flash, bootstrapping the entire module, effectively non-modifiable.
- Layer 1: Rewritable POST 1 and Miniboot 1, responsible for self-test and some card-level management functionality.

The upper two layers customize the operation of each individual device.

- Layer 2: System software. Supervisor-level code, including any system-provided device drivers, but excluding the startup stub.
- Layer 3: Application code, including userspace drivers, if any

1.4.5 Cryptographic engines

The architecture of 4767 and 4765 modules is comparable (Fig. 2). Internal access control between processors and code flash, communication channels etc. are logically quite similar.

Cryptographic engines available in both module models are the following:

1. TDES
2. AES
3. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
4. modular mathematics supporting modular multiplication (such as used as primitives of RSA, DSA, DH)
5. Hardware entropy source [note: conditioning, in both cases, is entirely software-based] [note: the hardware noise source is part of TOE environment.]

In addition to the above engines, the 4767 adds prime-field elliptic curve (EC) functionality hardcoded in the Andretta ASIC.

These cryptographic engines are implemented in the Otello ASIC and Rigolone FPGA on the 4765 module, and in the Andretta ASIC of the 4767 module. Those hardware items are therefore part of the TOE.

Differences between the two models are handled within EP11 code, and are not visible at TOE external interfaces (except with respect to performance). Firmware depends on engine-aware libraries abstracting away these differences (these libraries are already included in Segment 3 configurations).

1.5 Non-TOE hardware/software/firmware components

The TOE within its hosting HSM does not exist in isolation; it requires a set of conditions in its operating environment. Although the CryptoExpress PCIe cards can operate on various hardware platforms, this evaluation only considers the IBM zSeries mainframe platform that runs a Common Criteria evaluated version of the z/OS operating system.

HSMs hosting the TOE are typically accessed through PKCS#11 services of some host library such as the central cryptographic service provider of IBM mainframe operating systems, the *Integrated Cryptographic Service Facility (ICSF)* [ICS12].

The deployment, initialization and configuration of the TOE also requires a *Trusted Key Entry* (TKE) workstation to be present and attached to the zSeries mainframe and the z/OS system running on it. The TKE workstation thereby acts as the designated entity on which administrative tasks for the TOE are carried out by the system administrators, and on which site-specific settings such as compliance options can be selected conveniently. Documentation associated to the TKE workstation product provides guidance about its installation as well as the installation and configuration of the TOE [TKE16].

With the exception of signatures generated by the TKE workstation, the TOE expects no cryptographic functionality from the host. From the viewpoint of the TOE, the TKE workstation and the non-signing services that it provides, as well as the involved interfaces between abstraction layers (such as ICSF, the central cryptographic service provider of the z/OS operating system [ICS12]) are not considered trusted.

Reflecting the lack of trust in external entities, the TOE provides an end-to-end secured channel when it participates in state migration; digital signature covers the entire exported state and prevents modification by intervening parties. TKE administrative commands are similarly covered by digital signatures; communications between TKE and TOE instances are signed, and we expect no additional integrity protection from the (untrusted) entities on these paths.

To facilitate simpler compliance detection, domains report a condensed form of their control point setup as "compliance attributes", and are expected to be passed to applications/host administrators through ICSF queries or TKE displays. Note that host tooling, such as TKE consoles, may include single-button controls to select compliance [TKE16, Fig. 122].

While outside TOE scope, ICSF provides the user-visible, PKCS#11-compliant external interfaces corresponding to TOE-level EP11 ones. Since ICSF maps PKCS#11-using application calls to the TOE, most of its interaction is with non-administrative ("functional") TOE interfaces. ICSF also presents card and domain compliance settings as vendor-extended PKCS#11 attributes to PKCS#11 applications [ICS12, Table 10] or make it available to the entire system through logging facilities [ICS16, Displaying coprocessor hardware status].

2 Conformance claims

This Security Target and the TOE conform to version 3.1 (rev. 4) of the Common Criteria for Information Technology Security Evaluation with the following specific claims:

1. *CC Part 2,* extended
2. *CC Part 3,* conformant
3. Package conformant to EAL4.

This security target does not claim conformance to any protection profile.

3 Security Problem Definition

3.1 Assumptions

A.Key_Generation Key generation and import to the cryptographic module

Cryptographic keys generated by the IT environment and imported into the TOE are cryptographically strong for the intended key usage and have secure security attributes.

A.Audit_Analysis Analysis of audit trails

The TOE environment retrieves the audit records of the TOE and analyses them for security violations.

A.Availability Availability of keys

The TOE environment ensures the availability of cryptographic keys, key components, critical security parameters (CSPs) and key material.

Application note: *Sensitive state may be stored outside the TOE, on essentially untrusted hosts.* Sensitive state is encrypted and authenticated using a combination of AES/256 and HMAC-based signatures, using a MAC key derived from the controlling WK. Blobs are wrapped in an *Encrypt-then-MAC* structure [BN08, 1.2, 4.3] [Kra01, 4.1].

Application note: Non-sensitive objects MAY be authenticated by the same authentication infrastructure to allow authentication even without encryption. One application is authenticating attributes together with public keys.

Identifier	Threat statement
T.Compro_CSP	**Compromise of confidential CSP.** An attacker with enhanced-basic attack potential may compromise confidential CSP like secret keys, private keys or confidential authentication data, which enables attacks against the confidentiality or integrity of user data protected by these CSPs or the TSF using these CSPs as TSF data.
T.Modif_CSP	**Modification of integrity sensitive CSP.** An attacker with enhanced-basic attack potential may modify integrity sensitive CSP like permanent stored public keys and therefore compromise the confidentiality or integrity of user data protected by these CSPs or the TSF using these CSPs as TSF data.
T.Inf_Leakage	**Information leakage.** An attacker with enhanced-basic attack potential may observe and analyse any timing behaviour observable through the cryptographic boundary (i.e. including the external interfaces) of the TOE to get internal secrets (especially secret or private cryptographic keys) or confidential user data not intended for export. The information leakage may be inherent in the normal operation or caused by the attacker — such as by selection of specific input data, submitted through valid service calls.
T.Malfunction	**Malfunction of TSF.** An attacker with enhanced-basic attack potential may use a malfunction of the hardware or software, which is accidental in order to deactivate, modify, or circumvent security functions of the TOE to enable attacks against the integrity or confidentiality of the User data or the CSP.
T.Masquerade	**Masquerade authorized data source or receiver.** An attacker with enhanced-basic attack potential may masquerade as an authorized data source or receiver to perform operations that will be attributed to the authorized user or may gain undetected access to cryptographic module causing potential violations of integrity or confidentiality of the User data, the CSP or the TSF data.

Table 1: Threats

Application note: *The TOE in its evaluated configuration does not include firmware-update capabilities.*

A.Phys_Protection Physical protection

It is assumed that the IT environment provides the TOE with appropriate physical security, commensurate with the value of the IT assets protected by the TOE.

Application note: The assumption for physical protection is inherited from the TOEs operating environment, which is a z Systems mainframe running a z/OS operating system in its evaluated configuration. Note also that the HSM hosting the TOE comes with tamper-protection features that detect tampering and erase critical data when such tampering is detected. These tamper-protection features have not been subject to this evaluation.

3.2 Threats

The following threat agents are acknowledged by this security target:

1. host entities, capable of passive observation of the TOE without using functional TOE services, attempting to gain access to information or functions without attempting active compromise ("malicious observers," passive host entities attempting to query/access information not intended for them)
2. host entities, with explicitly allowed access, attempting to use, access or otherwise utilize TOE services and resources in a way which the TOE setup prohibits for them ("malicious,users allowed to access the TOE," those attempting to access services beyond what they are authorized to)

Table. 1 enumerates the threats considered by this security target.

Note that the definition of malicious observers includes all of the host, since the TOE provides many services without any authorization — including all queries. Since these services are intentionally widely available to facilitate transparent auditing, and access to them is is not restricted within the hosting HSM, we assume that all external entities are implicitly authorized to access such services. Therefore, when passive observers are included — which is already sufficient to exploit, as an example, side-channel attacks (**T.Inf_Leakage**, Information leakage) — all passive host entities are considered just as dangerous as authorized ones are. In other words, *our threat model includes malicious observers*.

Closely following the traditions of mainframe-related, stateless devices, *we mainly ignore threats related to denial-of-service* within the TOE. When deployed on enterprise platforms, the hosts may rate-limit or otherwise restrict overloading of individual HSMs more efficiently than devices themselves.

Note that since our assumptions specifically exclude firmware updates during the operation of EP11, *threats involving maliciously loaded software are not listed as applicable. This omission is intentional, it is not an oversight.*

Identifier	Organizational Security Policy statement
OSP.User_Data_Prot	**Protection of user data by cryptographic functions.** The cryptographic module will be used to protect the confidentiality or integrity or both of information represented by user data which may be get known or modified by an attacker. The IT system will ensure the availability of the user data and the cryptographic keys outside the cryptographic module.
OSP.I&A(Admin)	**Identification and authentication of administrative users.** All administrative users must be identified and authenticated prior to accessing any controlled resources *with the exception of read-only access to public objects*.
OSP.Access	**Access control of TOE functions.** The TOE must limit the extent of each user's abilities to use the TOE functions in accordance with the TSP.
OSP.Account(Admin)	**Accountability of (administrative) users** Administrative users of the TOE shall be held accountable for their actions within the system. Note that accountability is generally restricted to administrative functions (those changing HSM-internal state).
OSP.Roles	**Roles.** The authorized administrators and users shall have separate and distinct roles associated with them.
OSP.Endorsed_Crypto	**Endorsed cryptographic functions.** The TOE shall implement Endorsed cryptographic algorithms and Endorsed cryptographic protocols for the protection of the confidentiality or the integrity or both of the user data according to the organizational security policy **OSP.User_Data_Prot** and for the cryptographic key management according to the organizational security policy **OSP.Key_Man**. The cryptographic module must not provide any non-Endorsed cryptographic function.
OSP.Key_Man	**Cryptographic key management.** The CSP, cryptographic keys and cryptographic key components are assigned to cryptographic algorithms and protocols they are intended to be used with and the entities, which are allowed to use them.
OSP.Key_Personal	**Personal security for cryptographic keys.** The cryptographic keys shall be managed in such a way that their integrity and confidentiality cannot be compromised by a single person. Application note: Specific exceptions are allowed if the TOE is configured to allow control of cryptographic keys by single entities.

Table 2: Operational security policies

3.3 Organizational Security Policies

The organizational security policies of Table 2 are relevant in the context of this security target. *Since the TOE stores most sensitive data externally, and therefore relies on integrity protection of data passed through untrusted external components, data protection against highly capable attackers is critical.*

As a special case of personal security (**OSP.Key_Personal**), administrators *may* enable control of specific domains by single administrator keys. (As with all other administrative setup, this action is world-observable and auditable.) Therefore, the TOE allows single-administrative control of such keys, if this non-default capability is explicitly enabled.

While compatibility with the PKCS#11 standard prevents EP11 from entirely rejecting keys with problematic combinations of functionality — such as keys which are allowed to both un/wrap and en/decrypt — the TOE provides additional restrictions to prevent use of such keys. Therefore, the interpretation of **OSP.Key_Man** depends on attribute setup.

4 Security Objectives

4.1 Security Objectives, TOE

O.Endorsed_Crypto Endorsed cryptographic functions

The TOE shall provide Endorsed cryptographic functions and Endorsed cryptographic protocols to protect the user data as required by **OSP.User_Data_Prot** and for key management.

O.I&A Identification and authentication of administrative users

The TOE shall uniquely identify administrative users and verify the claimed identity of the user before providing access to any controlled resources with the exception of read access to public objects. The security functions for authentication of users shall have strength "enhanced-basic".

O.Roles Roles known to TOE

The TOE shall provide at least card and domain Administrator, and the User roles.

O.Control_Services Access control for services

The TOE shall restrict the access to its services, depending on the user role, to those services explicitly assigned to this role. Assignment of services to roles shall be either done by explicit action of an Administrator or by default.

O.Control_Keys Access control for cryptographic keys

The TOE shall restrict the access to the keys, key components and other CSP according to their security attributes. Cryptographic keys intended for the use with Endorsed cryptographic functions must not be used by any non-endorsed functions.

O.Audit Audit of the TOE

The TOE shall provide the capability to detect and create audit records of security relevant events associated with users.

O.Key_Export Export of cryptographic keys

The TOE shall export cryptographic keys with their security attributes. The cryptographic keys and their security attributes shall be protected in integrity. The TOE shall ensure the confidentiality of secret and private keys exporting them in encrypted form to authorized entities.

O.Key_Generation Generation of cryptographic keys by the TOE

The TOE shall generate cryptographic strong keys using Endorsed cryptographic key generation algorithms.

O.Key_Import Import of cryptographic keys

The TOE shall import keys with security attributes and verify their integrity. The TOE shall import secret or private keys in encrypted form or manually using split knowledge procedures only.

O.Key_Management Management of cryptographic keys

The TOE shall securely manage cryptographic keys, cryptographic key components and CSP. The TOE shall associate security attributes of the entity the key is assigned to and of the intended cryptographic use of the key. Assignment of the security attributes to the cryptographic keys, cryptographic key components and CSP shall be either done by explicit action of a Cryptographic Administrator or by default.

O.Key_Destruction Destruction of cryptographic keys

The TOE shall destruct in a secure way the keys cryptographic key components and other CSP on demand of authorized users or when they will not be used any more that no information about these keys is left in the resources storing or handling these objects before destruction.

Application note: The scope of **O.Key_Destruction** is limited to the TOE, but excludes externally stored replicas (which are outside TOE scope).

O.Check_Operation Check for correct operation

The TOE shall perform regular checks to verify that its components operate correctly. This includes integrity checks of TOE software, firmware, internal TSF data and keys during initial start-up, at the request of the authorised user, and at the conditions installation and maintenance.

O.Prevent_Inf_Leakage Prevent leakage of confidential information

The TOE shall prevent information leakage about secret and private keys and confidential TSF data outside the cryptographic boundary and unintended output confidential user information. The TOE shall resist attacks with enhanced-basic attack potential, which are based on information leakage.

4.2 Security Objectives, Operational Environment

OE.Key_Generation Key generation by IT environment

The IT environment shall ensure the cryptographic strength, the confidentiality and integrity of secret and private keys, the integrity and authenticity of public keys and correct security attributes if they are generated outside the TOE and imported into the TOE.

OE.Audit_Analysis Analysis of TOE audit data

The TOE environment reviews the audit trails generated and exported from the TOE to detect security violation and making authenticated users accountable for their actions related to the TOE. The administrator is responsible for configuration of the audit function and provision of the complete chain of exported audit trails.

OE.Personal Personal security

The card- or domain Administrator, User roles, and — if supported by the TOE — the Maintenance Personal role shall be assigned with distinct authorized persons. For manual key import at least two different authorized persons are assigned to cryptographic administrator role.

	OSP.User_Data_Prot	OSP.I&A	OSP.Access	OSP.Account	OSP.Roles	OSP.Endorsed_Crypto	OSP.Key_Man	OSP.Key_Personal	T.Compro_CSP	T.Modif_CSP	T.Inf_Leakage	T.Malfunction	T.Masquerade	A.Key_Generation	A.Audit_Analysis	A.Availability	A.Phys_Protection
O.I&A		X		X				X					X				
O.Control_Services			X														
O.Control_Keys			X					X	X	X			X				
O.Roles			X		X												
O.Audit				X													
O.Key_Export							X	X	X	X							
O.Key_Generation							X		X								
O.Key_Import							X	X	X	X							
O.Key_Management							X	X	X	X							
O.Key_Destruction							X		X								
O.Check_Operation										X		X					
O.Endorsed_Crypto	X					X											
O.Prevent_Inf_Leakage									X		X						
OE.Key_Generation							X		X					X			
OE.Audit_Analysis				X											X		
OE.Personal					X			X									
OE.Key_Availability	X															X	
OE.Phys_Protection																	X

Table 3: Security objective rationale

Application note: Manual key import is not supported in the evaluated configuration.

OE.Key_Availability Availability of cryptographic key and key material

The IT environment shall ensure the availability of the user data, cryptographic keys key components, CSP and key material.

OE.Phys_Protection Physical Protection

The IT environment deploying the TOE, as well as the HSM itself, shall resist physical attacks.

Application note: The TOE inherits any physical protection as an environmental feature of the HSM it is deployed in.

4.3 Security Objectives, rationale

The organisational security policy **OSP.User_Data_Prot** "Protection of user data by cryptographic functions" addresses the protection of the confidentiality or integrity or both of information represented by user data of the IT-system to be provided by the cryptographic module and the protection of availability of user data by the IT system. The security objective **O.Endorsed_Crypto** ensures that TOE provides Endorsed cryptographic functions to protect the user data as required by **OSP.User_Data_Prot**. The security objective for the IT environment **OE.Key_Availability** ensures that IT system protects the availability of the user data and the cryptographic keys outside the cryptographic module.

The organisational security policy **OSP.I&A** "Identification and authentication of users" and authentication of all users prior to accessing any controlled resources with the exception of public objects. This is directly ensured by the security objective **O.I&A**.

The organisational security policy **OSP.Access** "Access control of TOE functions" address the limitation of the extent of each users abilities to use the TOE functions in accordance with the TSP.

The security objective **O.Control_Services** requires the TOE shall restrict the access to its services, depending on the user role, to those services explicitly assigned to this role which are provided according to the security objective **O.Roles**. The security objective **O.Control_Keys** limits users ability to use the TOE functions to ensure the cryptographic security as part of the TSP.

The organisational security policy **OSP.Account** "Accountability of users" requires the users be held accountable for their actions within the system. The TOE security is required to establish the identity of the users by the objective **O.I&A** and to provide the capability to create audit records of security relevant events associated with users by the objective **O.Audit**. The security objective for the IT environment **OE.Audit_Analysis** ensures reviews of the audit trails generated and exported from the TOE making authenticated users accountable for their actions related to the TOE.

The organisational security policy **OSP.Roles** "Roles" addresses separate and distinct roles for authorized administrator, and users. The security objective **O.Roles** requires the TOE to implement them and the security objective **OE.Personal** requires the IT environment to use them.

The organisational security policy **OSP.Endorsed_Crypto** "Endorsed cryptographic functions" address the implementation of Endorsed cryptographic algorithms and Endorsed cryptographic protocols for the protection of the confidentiality or the integrity or both of the user data according to the organizational security policy **OSP.User_Data_Prot** and for the key management. This is ensured generally by the security objective **O.Endorsed_Crypto**.

The security objective **OSP.Key_Man** "Cryptographic key management" requires to manage and use the cryptographic keys as they are assigned to the entities, cryptographic algorithms and protocols. This OSP is implemented generally by the security objectives for the TOE **O.Key_Management** for secure key management and specifically for critical processes over the key life cycle by the security objectives **O.Key_Generation**, **O.Key_Import**, **O.Key_Export** and **O.Key_Destruction**. **OE.Key_Generation** ensures the cryptographic strength, the confidentiality and integrity of secret and private keys, the integrity and authenticity of public keys and correct security attributes if they are generated outside the TOE and imported into the TOE.

The organisational security policy **OSP.Key_Personal** "Personal security for cryptographic keys" addresses key management in a way that the integrity and confidentiality of key can not be compromised by a single person. This OSP is implemented generally by the security objectives **O.Key_Management** and **O.Control_Keys** for secure key management and use. Furthermore for critical processes, the security objectives **O.Key_Import**, **O.Key_Export** and **O.Control_Keys** enforce secure key import, key export and key usage. **O.I&A** ensures that the TOE uniquely identifies users and verifies the claimed identity of the user before providing access. **OE.Personal** requires assignment of roles to distinct authorized persons and that for manual key import at least two different authorized persons are assigned to cryptographic administrator role.

The threat **T.Compro_CSP** "Compromise of CSP" addresses the compromise confidential CSP which enables attacks against the confidentiality or integrity of user data and TSF data protected by these CSPs. The security objective **O.Control_Keys** requires the TOE to restrict the access to the keys, key components and CSP according to their security attributes. The security objective **O.Key_Management** ensures these security attributes are managed securely. The security objective **O.Key_Export** and **O.Key_Import** require the protection of secret or private keys in encrypted form or using split knowledge procedures for their export and import. The security objectives **O.Key_Generation** requires the TOE and the **OE.Key_Generation** requires the environment to generate cryptographic strong keys. **O.Key_Destruction** requires the secure destruction on demand of user. The security objective **O.Prevent_Inf_Leakage** requires the TOE to prevent information leakage about secret and private keys and confidential TSF data outside the cryptographic boundary.

The threat **T.Modif_CSP** "Modification of integrity sensitive CSP" address the modification of the integrity sensitive CSP which enables attacks against the confidentiality or integrity of user data or the TSF protected by these CSPs. The security objective **O.Control_Keys** requires the TOE to restrict the access to the keys, key components and CSP according to their security attributes.

The security objective **O.Key_Management** ensures these security attributes are managed securely. The security objective **O.Key_Export** and **O.Key_Import** require the protection of the integrity keys during their export and import. The security objective **O.Check_Operation** requires verification the integrity of CSP.

The security objective **O.Roles** requires the TOE to provide at least card and domain Administrators, and User roles, and Maintenance Personnel if the TOE supports maintenance functionality. The Administrator, User roles and Maintenance Personnel — if the TOE supports maintenance functionality — will be assigned to authorized distinct persons according to the security objective for the IT environment **OE.Personal**.

The threat **T.Inf_Leakage** "Information leakage" describes that an attacker may observe and analyse any operation timing through the cryptographic boundary (i.e. including the external interfaces) of the TOE to get internal secrets or confidential user data not intended for export. The protection against this threat is directly required by the security objective **O.Prevent_Inf_Leakage**.

The threat **T.Malfunction** "Malfunction of TSF" describes the use of a malfunction of the hardware or software in order to deactivate, modify, or circumvent security functions of the TOE to enable attacks against the integrity or confidentiality of the User data or the CSP. The security objective **O.Check_Operation** prevents this threat by regular checks verifying that TOE components operate correctly.

The threat **T.Masquerade** "Masquerade authorized data source or receiver" describes that an attacker may masquerade as an authorized data source or receiver to perform operations that will be attributed to the authorized user or gains undetected access to cryptographic module causing potential violations of integrity, or confidentiality. The security objective **O.I&A** requires the TOE to identify and authenticate the user before providing access to any controlled resources with the exception of public objects. The security objective **O.I&A** requires the security functions for authentication of users to have strength "enhanced-basic" to cover attacks with enhanced-basic attack potential as described in **T.Masquerade**.

The security objective **O.Control_Keys** restricts the access to the keys, key components and other CSP according to their security attributes (including Key entity).

The assumption **A.Key_Generation** "Key generation and import to the cryptographic module" deals with the cryptographic strength and secure security attributes of cryptographic keys generated by the IT environment and imported into the TOE. This assumption is directly and completely covered by the security objective for the IT environment **OE.Key_Generation**.

The assumption **A.Availability** "Availability of keys" describes that the IT environment ensures the availability of cryptographic keys and key material as ensured by the security objective for the IT environment **OE.Key_Availability**.

The assumption **A.Audit_Analysis** "Analysis of audit trails" addresses reading and analysis of audit records of the TOE as implemented by the security objective for the IT environment **OE.Audit_Analysis**.

The assumption **A.Phys_Protection** "Physical protection" addresses expectations of physical security of the environment deploying the TOE as implemented by the security objective for the IT environment **OE.Phys_Protection**.

5 Extended Components Definition

This Security Target does not define its own extended components. The requirement **FCS_RNG.1** "Random Number Generator" uses the extended component definition from [KS11, 3.1].

6 Security Requirements

Requirements in this section have been extracted from CC Part 2 providing functional requirements, and Part 3 contributing assurance requirements.

6.1 Security Functional Requirements

Key management (CKM) and related requirements

The following list of key types may be generated or imported to the TOE: AES [Nat01a], TDES [Nat99], RSA, EC (elliptic curve, prime field, NIST P-curves or BP prime-field elliptic curves [LM10]), DSA ([Nat13a]), DH, generic secret keys (a PKCS#11 abstraction, representing arbitrary sensitive bytes as raw data [PKC04, 12.7]).

FCS_CKM.1/AES Cryptographic key generation — AES

FCS_CKM.1.1/AES

The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *for AES* and specified cryptographic key sizes: *128, 192, 256 bit* that meet the following: **FIPS 197** *[Nat01a]*.

FCS_CKM.1/TDES Cryptographic key generation — TDES

FCS_CKM.1.1/TDES

The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *for TDES as specified by NIST SP800-67r1* and specified cryptographic key sizes: *168 bit* that meet the following: **NIST SP800-67r1** *[BB12, 3.2, 3.4.1]*

FCS_CKM.1/RSA Cryptographic key generation — RSA

FCS_CKM.1.1/RSA

TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *for RSA as specified by FIPS 186–4* and specified cryptographic key sizes: *2048, 3072, 4096 bits* that meet the following: **FIPS 186–4** *[Nat13a, 5.1, B.3.1]*.

Application note: RSA private keys constructed from primality search specified by [Nat13a, B.3.1] are also compliant with PKCS1 [JK03, 3.2] (which does not specify details of prime-candidate enumeration in comparable detail, but specifies subsequent use/encoding of keys). "PKCS1" key generation of specific key sizes — when compatible with [Nat13a, B.3.1] sizes — is simply mapped to the specific prime-candidate enumeration method of [Nat13a, B.3.1].

FCS_CKM.1/DSA Cryptographic key generation — DSA

FCS_CKM.1.1/DSA

TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *for DSA as specified by FIPS 186–4* and specified cryptographic key sizes *DSA: 2048 or 3072 bits* that meet the following: **FIPS 186–4** *[Nat13a, 4.4.1, B.1]*.

FCS_CKM.1/EC Cryptographic key generation — EC

FCS_CKM.1.1/EC

TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *elliptic curve keypair generation* with specified cryptographic key sizes: *192, 224, 256, 320, 384, 512 or 521 bits*, that meet the following:

According to FIPS 186–4, B.4.2, "Keypair generation by testing candidates" over low-cofactor prime-field curves [Nat13a, 6.1.1].

Supported elliptic curves are:

- *EC/NIST: NIST P-curves P-192, P-224, P-256, P-384, P-521 [Nat13a, D.1.2]*
- *Brainpool (prime-field) curves: BP192r1/t1, BP224r1/t1, BP256r1/t1, BP320r1/t1, BP384r1/t1, BP512r1/t1 [LM10, 3]*

FCS_CKM.2/Import Cryptographic key distribution — Import

FCS_CKM.2.1/Import

The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method *key entry* that meets the following: *keys can only be imported in encrypted form, optionally using split knowledge procedures for administrative keys using administrator-selected thresholds and keypart counts*

Application note: Key import is in encrypted form, optionally under split-knowledge procedures [BSI08, 6.1.1]. The TOE does not support clearkey key transport. Key encryption uses AES with a card-specific 256-bit key, implementing key wrapping algorithm AES-KW1 according to RFC 3394 and RFC 5649 [HD09, HS02].

FCS_CKM.2/Export Cryptographic key distribution — Export

FCS_CKM.2.1/Export

The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method *key export* that meets the following: *keys can only be exported in encrypted form, optionally using split knowledge procedures for administrative keys using administrator-selected thresholds and keypart counts*

Application note: Key import — and export, if applicable — are in encrypted form, optionally under split-knowledge procedures (Fig. 5) [BSI08, 6.1.1] The TOE does not support clearkey key transport.

Application note: Functional key-transport, as described by PKCS#11 `UnwrapKey()` and `WrapKey()` calls, transports keys as single parts in encrypted form. While EP11 replaces the original — unauthenticated — PKCS#11 format with integrity-protected and signed formatting, and always includes attributes with keys — both are incompatible changes — it retains the single-pass nature of key transport.

Application note: Administrative key transport, unless single-part transport is enabled, uses split-knowledge procedures, individually encrypting keyparts. Split keys use Shamir key-sharing [Sha79] with administrator-selected thresholds and keypart counts.

Application note: All key import and export is authenticated through digital signatures or symmetric MACs, depending on context.

Application note: Sensitive state stored on the host is not treated as "import" or "export" in the PKCS#11 sense, as this object storage is opaque to PKCS#11 (it does not cross PKCS#11 `Un/WrapKey()` calls). For the purposes of these security functional requirements, blob-related state import and export qualifies as such, and fulfills the requirements.

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 *overwriting plaintext with zeros when the key is to be destroyed, or when a transient copy is no longer used* that meets the following: **FIPS 140–2** *[FIP01, 4.7.6]*.

Application note: Keys are stored in regular RAM or in battery-backed RAM. Objects on the heap or stack, when released, are overwritten with zeroes.

Application note: Specific destruction methods as may be required for flash memory or EEPROM are not required here. The TOE has a function for emergency erasing keys in case tampering is detected, but since this function is triggered by hardware and since this hardware is considered to be part of the IT environment in this evaluation, the function for emergency erasing keys is not assessed in this evaluation.

FCS_COP.1 Cryptographic operation

FCS_COP.1/AES Cryptographic operation — AES

FCS_COP.1.1/AES

The TSF shall perform *encryption and decryption* in accordance with a specified cryptographic algorithm *AES in ECB or CBC modes* with cryptographic key sizes *128, 192, and 256 bits* that meet the following: **FIPS 197** *[Nat01a]*, **NIST SP800-38A** *[Nat01b]*.

FCS_COP.1/TDES Cryptographic operation — TDES

FCS_COP.1.1/TDES The TSF shall perform *encryption and decryption* in accordance with a specified cryptographic algorithm *TDES in ECB or CBC modes* with cryptographic key size *168 bit* that meet the following: **NIST SP800-67r1** *[BB12, 3.2]*, **NIST SP800-38A** *[Nat01b]*.

FCS_COP.1/RSA Cryptographic operation — RSA

FCS_COP.1.1/RSA

The TSF shall perform *digital signature generation and verification, encryption and decryption* in accordance with a specified cryptographic algorithm *RSA* and cryptographic key sizes *2048, 3072, 4096 bits* that meet the following:

Encryption:

- *RSAEP of RFC 3447 (PKCS1 v2.1) [JK03, 5.1.1]*
- *According to RSAES-PKCS1-V1_5-ENCRYPT of RFC 3447 (PKCS v2.1) [JK03, 7.2.1]*
- *According to RSAES-OAEP-ENCRYPT of RFC 3447 (PKCS v2.1) [JK03, 7.1.1]*

Decryption:

- *RSAEP of RFC 3447 (PKCS1 v2.1) [JK03, 5.1.2]*
- *According to RSAES-PKCS1-V1_5-DECRYPT of RFC 3447 (PKCS v2.1) [JK03, 7.2.2]*
- *According to RSAES-OAEP-DECRYPT of RFC 3447 (PKCS v2.1) [JK03, 7.1.2]*

Signature generation:

- *RSASP1 of RFC 3447 (PKCS1 v2.1) [JK03, 5.2.1]*
- *According to RSAES-PKCS1-V1_5-SIGN of RFC 3447 (PKCS v2.1) [JK03, 8.2.1]*
- *According to RSAES-PSS-SIGN [JK03, 8.1.1]*

Signature verification:

- *RSAVP1 RFC 3447 (PKCS1 v2.1) [JK03, 5.2.2]*
- *According to RSAES-PKCS1-V1_5-VERIFY of RFC 3447 (PKCS v2.1) [JK03, 8.2.2]*
- *According to RSAES-PSS-VERIFY of RFC 3447 (PKCS v2.1) [JK03, 8.1.2]*

Application note: In RSA signature modes, hash functions SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 are usable. PSS/OAEP use of hash functions restricts message and mask generation function (MGF) choices to identical functions—one may not select mismatched message-hash/MGF pairs.

Application note: PKCS#11 terminology of "key un/wrapping" excludes security attributes, describes only raw — symmetric — keys being encrypted by asymmetric ones [PKC04, 11.14]. This usage corresponds to the raw en/decryption primitive of [JK03, 8, 7.1], even if the description there does not specifically mention un/wrapping.

FCS_COP.1/ECDSA Cryptographic operation — ECDSA

FCS_COP.1.1/ECDSA

The TSF shall perform *ECDSA signature generation or verification* in accordance with a specified cryptographic algorithm *Elliptic Curve Digital Signature Algorithm (ECDSA)* with specified cryptographic key sizes: *192, 224, 256, 320, 384, 512 or 521 bits*, that meet the following:

ECDSA signature generation: According to ANSI X9.62-2005, section 7.3; ECDSA signature verification: ANSI X9.62-2005, section 7.4.1 (verification with a public key) [Ame05, 7.3, 7.4.1].

The underlying hash functions SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 are supported.

Supported elliptic curves are:

- *EC/NIST: NIST P-curves P-192, P-224, P-256, P-384, P-521 [Nat13a, D.1.2]*
- *Brainpool (prime-field) curves: BP192r1/t1, BP224r1/t1, BP256r1/t1, BP320r1/t1, BP384r1/t1, BP512r1/t1 [LM10, 3]*

FCS_COP.1/ECDH Cryptographic operation — ECDH

FCS_COP.1.1/ECDH

The TSF shall perform *EC Key Agreement* in accordance with a specified cryptographic algorithm *Elliptic Curve Diffie-Hellman Key Exchange (ECDH)* with specified cryptographic key sizes: *192, 224, 256, 320, 384, 512 or 521 bits*, that meet the following:

EC Key Agreement: According to ANSI X9.63-2001, section 5.4.1 [Ame01, 5.4.1] (also [Nat13b, 5.7.1.2]).

Supported elliptic curves are:

- *EC/NIST: NIST P-curves P-192, P-224, P-256, P-384, P-521 [Nat13a, D.1.2]*
- *Brainpool (prime-field) curves: BP192r1/t1, BP224r1/t1, BP256r1/t1, BP320r1/t1, BP384r1/t1, BP512r1/t1 [LM10, 3]*

Application note: The TOE provides ECDH primitives for key agreement, but does not procedurally distinguish between *static* and *ephemeral* keys. Therefore, we may not identify the specific instance used from [Nat13b, 6.1–6.3], only the base ECDH primitive [Nat13b, 5.7.1.2].

FCS_COP.1/DSA Cryptographic operation — DSA

FCS_COP.1.1/DSA

The TSF shall perform *digital signature generation and verification* in accordance with a specified cryptographic algorithm *DSA* and cryptographic key sizes *P=2048, N=224 or 256; P=3072, N=256* that meet the following: **FIPS 186–4** *[Nat13a]*.

Application note: The hash-integrated versions may use hash functions SHA-224 and SHA-256 as underlying hash function (and corresponding secret-key bitcount). Note that hash functions beyond SHA-1 have not been standardized by the original PKCS#11 [PKC04] in 2009, and have been deprecated/prohibited by subsequent algorithm-strength restrictions.

Application note: The TOE in the evaluated configuration restricts parameter choices to the standard-predefined possibilities. Other combinations MAY NOT be enabled in the TOE; additional restrictions MAY further reduce the choices the TOE supports.

FCS_COP.1/SHA Cryptographic operation — SHA

FCS_COP.1.1/SHA

The TSF shall perform *secure hashing* in accordance with a specified cryptographic algorithm *SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256*, and cryptographic key sizes *(not applicable)* that meet the following: **FIPS 180–4** *[Nat12]*.

FCS_COP.1/Backup_Enc Cryptographic operation — Encryption of Backup data

> **FCS_COP.1.1/Backup_Enc** The TSF shall perform *encryption and decryption* in accordance with a specified cryptographic algorithm *AES in CBC mode* with cryptographic key size *256 bit* that meet the following: **FIPS 197** *(AES) and* **NIST SP800-38A** *(CBC mode) [Nat01a]*.

Application note: in addition to the AES/CBC transport-key encryption, Individual keyparts are encrypted through RSA or ECIES (see **FCS_CKM.1.1/EC**, **FCS_CKM.1.1/EC**).

FCS_COP.1/Backup_Int Cryptographic operation — Backup integrity protection

FCS_COP.1.1/Backup_Int The TSF shall perform *calculation and verification of cryptographic checksums* in accordance with a specified cryptographic algorithm *SHA-256, with RSA signatures* and cryptographic key sizes *4096-bit RSA* that meet the following: **FIPS 180-4** *[Nat12]*, **FIPS 186-4** *[Nat13a]*

Application note: the signature is the final section in backup data, signing the host-visible form of serialized state (i.e., non-sensitive data plus ciphertext). Backup RSA signatures use ANSI x9.31 padding and SHA-256 as a hash function.

Application note: in addition to signatures on backup state, individual parts of the migrated state are authenticated as individual administrative commands.

FCS_RNG.1 Generation of random numbers

FCS_RNG.1.1 The TSF shall provide a *[deterministic]* random number generator that implements:

DRG.3.1 If initialized with a random seed, from its internal hardware noise source. The internal state of the RNG shall *[have a minimum of 256 bits of entropy]*.

DRG.3.2 The RNG provides forward secrecy.

DRG.3.3 The RNG provides backward secrecy even if the current internal state is known.

FCS_RNG.1.2 The TSF shall provide random numbers that meet:

DRG.3.4 The RNG, initialized with a random seed *of 256 bits from its internal hardware noise source*, generates output for which 2^{34} strings of bit length 128 are mutually different with probability $P > 1\text{-}2^{-16}$.

DRG.3.5 Statistical test suites cannot practically distinguish the random numbers from output sequences of an ideal RNG. The random numbers must pass test procedure A **of AIS 31 [KS11, 2.4.4.1]** *and no other test suites*.

Application note: The DRNG is implemented as a `Hash_DRBG` using SHA-256 as its hash function according to [BK15, 10.1.1].

Application note: The DRNG is reseeded with 256 bits of fresh entropy after a maximum of 1 MB (2^{23} bits) output has been taken.

User identification (FIA)

FIA_ATD.1 User attribute definition

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

1. *Identity (for Administrators and Identified users)*
2. *Role (for Administrators)*
3. *Reference authentication data (i.e., certificates, for Administrators)*
4. *Session PIN (for Identified users)*

Application note: Administrative operations involving identities are authenticated cryptographically, relying on administrator public keys. Such calls index administrators through public-key hashes, and assign roles implicitly, as part of command selection.

Application note: The session PIN is used as a parameter for the function protecting the confidentiality and integrity of session bound objects. Without the correct specification of the session PIN the TOE will reject those objects when imported to the TOE for use with TOE operations.

FIA_UID.1 Timing of identification

FIA_UID.1.1 The TSF shall allow *the use of CSP objects not bound to a session, query commands* on behalf of the user to be performed before the user is identified.

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

FIA_UAU.1 Timing of authentication

FIA_UAU.1.1 The TSF shall allow *access to non-administrative functions and query commands unless they are prohibited by the domain control point access control policy* on behalf of the user to be performed before the user is authenticated.

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

Application note: as described under FIA_ATD.1.1, authentication is implicit, and coincides with use of keys in the same call sequence. Identification, where applicable, always precedes functional operations.

Application note: Unidentified users: non-administrators submitting requests which do not use session-bound objects, are not authenticated.

FIA_USB.1/GE User-subject binding — General

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

1. *Identity (for Administrators and Identified users)*
2. *Role (for Administrators)*

Application note: In addition to user security attributes, requests are also associated with domain-controlling keys implicitly, inferred from the domain targeted by the request (and are referenced by a corresponding indicator in host-supplied state). During subsequent processing, after cross-checking, these keys are checked and managed together with the Identity/Role of the request submitter.

FIA_USB.1.2/GE The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of users: the *initial role of the user is Unidentified user*.

In addition to user security attributes, except for card level administrative commands, the domain key is associated with the subject based on the Domain identifier passed in with the request. Domain keys are associated implicitly, inferred from the domain targeted by the request. During subsequent processing, after cross-checking, domain-related attributes are checked and managed together with the Identity/Role of the request submitter.

FIA_USB.1.3/GE The TSF shall enforce the following rules governing changes to the user security attributes associated with subjects acting on the behalf of users:

1. *the subject attribute Role shall be changed from Unidentified user to Identified user on presentation of a valid PIN Blob;*
2. *after successful authentication the subject attribute Role shall be changed to the Administrator role associated with the certificate used for authentication if the command is an administrative command requiring a valid signature;*

FIA_USB.1/IU User-subject binding — User

FIA_USB.1.1/IU The TSF shall associate the following user security attributes with subjects acting on the behalf of that *identified* user: *session PIN*.

FIA_USB.1.2/IU The TSF shall enforce the following rules on the initial association of user security attributes with subjects acting on the behalf of identified users: *the session PIN is initially provided with the `Login` function and the session token provided in return by the `Login` function needs to be provided for every operation that creates or uses a CSP that is bound to that session*.

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

FIA_AFL.1 Authentication failure handling

FIA_AFL.1.1 The TSF shall detect when *[any number of]* unsuccessful authentication attempts occur related to *administrative commands requiring authentication*.

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts has been *[met]*, the TSF shall *not perform the requested operation*.

Application note: Since the TOE is stateless, administrators are authenticated with every command they issue.

Application note: Since authentication is cryptographically assisted, and unsuccessfully authenticated requests are rejected, the TOE does not react to unsuccessful attempts other than rejecting requests. Conversely, no upper limit on unsuccessful authentication is defined.

Application note: As a side effect of cryptographic authentication, trivial denial-of-service attempts to invalidate any valid, authenticated entity are prevented, by construction.

Protection of user data (FDP)

FDP_ACC.1/CP Subset access control — Control Points

FDP_ACC.1.1/CP The TSF shall enforce *domain control point access control policy* on *Users as subjects and functions as objects* and *all subjects and the objects and functions affected by a control point*

FDP_ACF.1/CP Security attribute based access control — Control Points

FDP_ACF.1.1/CP The TSF shall enforce the *domain control point access control policy* to objects based on the following: *attributes: control points defined for the domains of the module; subjects: all submitted requests; objects: keys or state included in submitted requests*.

FDP_ACF.1.2/CP The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: *a restriction defined by a control point becomes active when the related control point bit is set for the active domain; in this case, the restriction defined by the control point is enforced for all subjects*.

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

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

FDP_ACC.2/OSA Complete access control — Object Security Attribute

FDP_ACC.2.1/OSA The TSF shall enforce the *object security attribute access control policy* on *users as subjects and key objects* and all operations among subjects and objects covered by the SFP.

FDP_ACC.2.2/OSA 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/OSA Security attribute based access control — Object Security Attribute

FDP_ACF.1.1/OSA The TSF shall enforce the *[object security attribute access control policy]* to objects based on the following: *[attributes: security attributes of key objects, subjects: sessions identified by requests which include keys or state bound to session identifiers]*.

FDP_ACF.1.2/OSA The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: *an operation is allowed if no attribute in the security attribute list prohibits the operation*.

Application note: Object attributes are listed in section 8.3.2 (page 47); most are implemented as defined by the PKCS#11 standard.

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

FDP_ACF.1.4/OSA The TSF shall explicitly deny access of subjects to objects based on the following additional rules: *[export of attribute-bound key objects without their attributes is denied, changing the* attribute-bound *attribute of objects is prohibited]*.

FDP_ACC.2/SO Complete access control — Session Objects

FDP_ACC.2.1/SO The TSF shall enforce the *session object access control policy* on *subjects: sessions referenced by requests (and Identified users, if session identification is successful); objects: keys or state bound to sessions* and all operations among subjects and objects covered by the SFP.

FDP_ACF.1/SO Security attribute based access control — Session Objects

FDP_ACF.1.1/SO The TSF shall enforce the *session object access control policy* to objects based on the following: *subject: session identifiers that are created upon session creation (`Login()` command), objects: keys or state bound to session identifiers if the request includes such state, attributes: presence or absence of the referenced session, indicating permission or prohibition, if the request includes keys or state bound to session identifiers*.

Application note: Session identifiers are created using the PIN entered by the user and a nonce generated by the TOE. The session identifier is a MAC value, calculated over the PIN and the nonce. Since session objects are also encrypted with the WK of the domain, session objects cannot be used outside the domain they have been created in.

Application note: When a `Login()` command is called by a user, a PIN blob is created which contains the session identifier. When calling functions that intend to create session objects, the call to the function needs to include the PIN blob as part of the parameter list.

Application note: When supplying session-bound object, sufficient information is included in the object-embedded session field to identify the controlling session, but not enough to reconstruct the originating PIN blob.

FDP_ACF.1.2/SO The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: *objects: session objects can only be accessed — used — within the domain where the session identifier was created, when the session has not been terminated, and when the related session identifier is provided in the function call, subjects: sessions associated with session identifiers when the function call uses state bound to sessions*.

FDP_ACF.1.3/SO The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: *[objects created without specifying a session identifier are considered public objects and can be accessed (used) by anyone]*.

FDP_ACF.1.4/SO

The TSF shall explicitly deny access of subjects to objects based on the following additional rules: *[when the WK key of the domain changes when a session in the domain is still active, access to previously created session objects is no longer possible]*.

FDP_ITC.2 Import of user data with security attributes

FDP_ITC.2.1 The TSF shall enforce the *domain access control policy* when importing user data, controlled under the SFP, from outside of the TOE.

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

FDP_ITC.2.3 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 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 The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TOE: *when the integrity check for imported user data fails, data is rejected*.

FDP_ETC.2 Export of user data with security attributes

FDP_ETC.2.1 The TSF shall enforce the *domain access control policy* when exporting user data, controlled under the SFP(s), outside of the TOE.

FDP_ETC.2.2 The TSF shall export the user data with the user data's associated security attributes.

FDP_ETC.2.3 The TSF shall ensure that the security attributes, when exported outside the TOE, are unambiguously associated with the exported user data.

Application note: see section 7.1.2 ("Transport of security attributes").

FDP_ETC.2.4 The TSF shall enforce the following rules when user data is exported from the TOE: *none*

FDP_IFC.2 Complete information flow control

FDP_IFC.2.1 The TSF shall enforce the *domain access control policy* on *information: all requests submitted to the TOE except card level administrative commands; subjects: sessions associated with requests when the latter include state bound to session identifiers; attributes: domain-specific CPs (7.1.6)* and all operations that cause that information to flow to and from subjects covered by the SFP.

FDP_IFC.2.2 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.

FDP_IFF.1 Simple security attributes

FDP_IFF.1.1 The TSF shall enforce the *domain access control policy* based on the following types of subject and information security attributes: *subjects: all requests except card-level administrative commands, information security attribute: domain identifier*.

FDP_IFF.1.2 The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: *all requests except card-level administrative commands must include a domain identifier. All objects exported will be encrypted with the wrapping key of that domain. They can not be used with requests that contain a different domain identifier*.

FDP_IFF.1.3 The TSF shall enforce the *no additional rules*.

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: *no additional rule*.

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: *no additional rule*.

Application note: This policy specifies that objects created by the TOE are always bound to a domain. This is achieved by using different domain wrapping keys to protect the confidentiality and integrity of those objects before exporting them to the host system.

FPT_ITI.1 Inter-TSF detection of modification

FPT_ITI.1.1 The TSF shall provide the capability to detect modification of all TSF data during transmission between the TSF and another trusted IT product which is another instance of the TOE within the following metric: *validation of digital signatures and hash-based verification patterns*.

FPT_ITI.1.2 The TSF shall provide the capability to verify the integrity of all TSF data transmitted between the TSF and another trusted IT product and perform *rejection of the data* if modifications are detected.

Application note: The TSF provides the capability to migrate keys or state information from one instance of the TOE to another one. This migration is controlled through authenticated administrative actions both on the exporting instance of the TOE as well as on the importing instance of the TOE. Exporter and importer instance of the TOE will first negotiate a transport wrapping key which is used to encrypt key material transported between the two instances of the TOE. Exported key material is encrypted using this key.

Application note: Commands to export and import must be signed by the configured number of administrators. The transport wrapping key may also be exported in several key parts, requiring several administrative users to co-operate during the negotiation of the wrapping key between the source and the target instance of the TOE.

FPT_STM.1 Reliable time stamps

FPT_STM.1.1 The TSF shall be able to provide reliable timestamps.

FDP_RIP.2 Full residual information protection

FDP_RIP.2.1 The TSF shall ensure that any previous information content of a resource is made unavailable upon the *[allocation of the resource to OR deallocation of the resource from]* all objects.

Application note: Objects released within the TOE, including deallocated persistent structures and transient memory regions, are wiped before they are released. During allocation, as a redundant security measure, newly allocated memory is explicitly cleared before use.

Application note: Since TOE operation is mainly stateless, other than state changes to global structures, most memory wiping impacts transient stack structures allocated during request-response processing. Changes to global structures update both heap-resident and persistent copies.

Security audit (FAU)

FAU_GEN.1 Audit data generation

FAU_GEN.1.1 The TSF shall be able to generate an audit record of the following auditable events:

(a) start-up and shutdown of the audit functions;

(b) all auditable events for the *[not specified]* level of audit; and

(c) completion of selftests

(d) import of keys or key components

(e) export of keys or key components

(f) destruction of TOE-resident keys

(g) recovery from detected failures

(h) administrative commands changing state

FAU_GEN.1.2 The TSF shall record within each audit record at least the following information:

(a) Date and time of the event, type of event, subject identity, and the outcome (success or failure) of the event; and

(b) For each audit event type, based on the auditable event definitions of the functional components included in the ST,

- *sequence number of the audit event*
- *audit-instance identifier of the event log*
- *identifier of the key(s) involved (where applicable)*
- *TOE-generated salt*

Application note: see section 7.1.9 ("Audit")

FAU_SAR.1 Audit review

FAU_SAR.1.1 The TSF shall provide *all host entities* with the capability to read *all audit data* from the audit records.

Application note: Reading audit records does not remove them from TOE-maintained history; therefore, providing read access to audit events does not allow the host to compromise event history through read-only access.

FAU_SAR.1.2 The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

Application note: Data structures — i.e., formatting and sub-fields interpretation — of audit records are fully documented as part of the external EP11 interface specification.

FAU_STG.1 Protected audit trail storage

FAU_STG.1.1 The TSF shall protect the stored audit records in the audit trail from unauthorised deletion.

FAU_STG.1.2 The TSF shall be able to *[prevent]* unauthorised modifications to the stored audit records in the audit trail.

Application note: Audit records, inserted into a non-malleable hash chain, are public information, accessible by crypto officers and administrators. The integrity protection prevents the hash chain from modification or deletion of already-logged entries.

See also: **FAU_STG.4**.

FAU_STG.4 Prevention of audit data loss

FAU_STG.4.1 The TSF shall *[overwrite the oldest stored audit records]* and *no other action* if the audit trail is full.

Application note: Since the TOE-internal audit history is sized conservatively; past items need not be overwritten before they may be saved by the host.

Management of TSF and protection of TSF data (FMT)

FMT_SMF.1 Specification of Management Functions

FMT_SMF.1.1 The TSF shall be capable of performing the following management functions: *(1) management of security functions behaviour, (2) management of reference authentication data, (3) management of security attributes of cryptographic keys, cryptographic key components and CSP.*

Application note: for related management functionality, see (1) **FMT_MTD.1.1/CP** (2) **FMT_MTD.1/Admin** and **FMT_MTD.1/User**, and (3) **FMT_MSA.1/Key_Man_1**, **FMT_MSA.1/Key_Man_2**, **FMT_MSA.3**.

FMT_SMR.1 Security roles

FMT_SMR.1.1 The TSF shall maintain the roles:

- *Administrator*

  - *Card-level administrator*
  - *Domain administrator*
- *Identified user*
- *Unidentified user*

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

Application note: roles correspond to the following PKCS#11 entities, and additional ones defined by the TOE for its necessary roles not representable by standard PKCS#11 ones:

1. Administrator: authenticated host entities capable of changing administrative setup
2. Identified user: user submitting a PKCS#11 request that uses a session-bound object within a session
3. Unidentified and unauthenticated user: any other host entity

Application note: Association with the host entity originating the request is implicit based on context.

Application note: The TOE does not support a Maintenance role.

FMT_MTD.1/Admin Management of TSF data — Administrator

FMT_MTD.1.1/Admin The TSF shall restrict the ability to *[create, clear and delete]* the *Reference Authentication Data* to *Administrator*.

Application note: Management of administrative identities — i.e., certificates — and corresponding attributes is performed through authenticated, administrative commands, not available to other users, or an insufficient number of cooperating administrators.

FMT_MTD.1/User_1 Management of TSF data — User identification

FMT_MTD.1.1/User_1 The TSF shall restrict the ability to *[define]* the *Reference Identification Data* to *the user for their own Reference Session Identification Data*.

Application note: Management of sessions, the only TOE-internally retained user identification data is available to users only after verifying proof-of-possession (i.e., removal of session data requires cooperation of the originating user). Administrators may forcibly terminate a session, but not identify as an authenticated user.

FMT_MTD.1/User_2 Management of TSF data — User de-identification

FMT_MTD.1.1/User_2 The TSF shall restrict the ability to *[delete]* the *Reference Identification Data* to *the user* **for their own Reference Session Identification Data**.

FMT_MTD.1/CP Management of TSF data — Control Points

FMT_MTD.1.1/CP The TSF shall restrict the ability to *[set or clear]* the *domain access-control points* to *administrator*.

FMT_MSA.1/Key_Man_1 Management of security attributes — Unmodifiable attributes

FMT_MSA.1.1/Key_Man_1 The TSF shall enforce the *object security attribute access control policy* to restrict the ability to *change default and modify* the security attributes *Identity of the key, Key entity, Key type, Key validity time period, Identity of the key component, Key entity of the key component, Key entry method, Identity of the CSP, CSP usage type* to *no role*.

FMT_MSA.1/Key_Man_2 Management of security attributes — Modifiable attributes

FMT_MSA.1.1/Key_Man_2 The TSF shall enforce the *object security attribute access control policy* to restrict the ability to *[modify]* the security attributes *Key usage type, key/state access control rules* to *Identified user*.

Application note: once created, the following attributes are not modifiable by any TOE-accessing entity:

- key, key component, or CSP identity
- key or CSP type
- key validity (which is not interpreted by the TOE)
- key or CSP entity (which is not interpreted by the TOE)
- key entry method (i.e., the "LOCAL" attribute)
- non-key CSP usage type (all non-key CSPs have predefined, non-modifiable usage patterns, such as: Digest states are only accepted by `C_Digest...` services)
- CSP access control rules

Application note: the following attributes are modifiable by the controlling owner, with standard restrictions removing subsequent capability of further modification:

- key usage type, within those allowed for the key type (such as: subsequently removing the capability to SIGN/WRAP etc., for keys where multiple algorithm types may be supported)
- key/state access control rules — possibly managed in the TOE, not the key itself — such as removing extractability from an originally EXTRACTABLE key

Application note: For some attributes, further restrictions may be added to prevent modification by any host entity, including the controlling user or administrator (i.e., effectively switch some state to NON-MODIFIABLE to potentially anyone). An intermediate restriction level, RESTRICTABLE is available to prohibit addition/activation, but still support deactivation/removal, of capabilities. While modification is allowed for the controlling user, restrictions may be imposed by other entities (Fig. 4).

Application note: Queries are available to every user capable of issuing card requests. Queries related to host-resident objects are implicitly restricted to users allowed to access those objects; no query modifies the keys/state it is accessing.

FMT_MSA.2 Secure security attributes

> **FMT_MSA.2.1** The TSF shall ensure that only secure values are accepted for *Identity of the key, Key entity, Key type of the key, Key usage type, Key access control rules, Key validity time period, Identity of the key component, Key entity of the key component, Key entry method, Identity of the CSP, CSP usage type, CSP access control rules*.

Application note: invalid host-supplied attributes are rejected during object creation — including key generation — and during any modification attempt, even authorized ones. Insecure attributes or attribute combinations are marked invalid when the applicable CP etc. setup rejects them (Fig. 4).

Application note: When an imported object with inconsistent attributes is encountered, even if properly authenticated, the object is rejected as invalid.

FMT_MSA.3 Static attribute initialisation

> **FMT_MSA.3.1** The TSF shall enforce the *object security attribute access control policy, session object access control policy* to provide *[restrictive]* default values for security attributes that are used to enforce the SFP.

> **FMT_MSA.3.2** The TSF shall allow *no role* to specify alternative initial values to override the default values when an object or information is created.

Application note: most attributes require explicit specification by the originator; prudent defaults are provided for several values if not specified. System-wide defaults are read-only, and not influenced by any attribute selection (which are only applied to particular objects).

Application note: The TOE may enable additional checks, rejecting previously generated objects as insufficiently restrictive, as dynamic administrative actions (such as when activating additional usage-control restrictions).

TSF protection (FPT)

FPT_TST.1 TSF testing

FPT_TST.1.1 The TSF shall run a suite of self tests *[during initial start-up OR at the request of any authorised user]* to demonstrate the correct operation of *[cryptographic primitives of the TSF]*.

FPT_TST.1.2 The TSF shall provide authorised users with the capability to verify the integrity of *[TSF data in persistent storage]*.

Application note: all persistently stored internal data is integrity-checked through an embedded SHA-256 hash [Nat12, 6.2], which is transparent to the rest of the TOE. When such data is made available, it has been implicitly verified by the TOE.

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

FPT_TDC.1 Inter-TSF basic TSF data consistency

FPT_TDC.1.1 The TSF shall provide the capability to consistently interpret *security attributes of cryptographic keys, key components and CSP* when shared between the TSF and another trusted IT product.

Application note: In addition to administrative authentication of the control and data-flow of inter-TSF sharing, the receiving instance also verifies signature on the data, and consistency of all imported components.

FPT_TDC.1.2 The TSF shall, during import, use *the encoded key attributes after it has verified their integrity* when interpreting TSF data from another trusted IT product.

Application note: see **FPT_TDC.1.1**. Note that state import is atomic. Any failure reverts to the original state, if applicable, or forces zeroization of the receiving module (if recovery is impossible due to unexpected infrastructure failures).

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: *self test fails*.

Application note: selftest failures prohibit further operations, generating log entries if still possible. Recovery actions, such as resetting the module through host infrastructure commands, may be attempted by the host.

6.2 Security Assurance Requirements

The EAL4 was chosen to permit a developer to gain maximum assurance from positive security engineering based on good commercial development practices which, though rigorous, do not require substantial specialist knowledge, skills, and other resources. EAL4 is applicable in those circumstances where developers or users require high levels of independently assured security in conventional commodity TOEs and are prepared to incur security specific engineering costs.

The evaluation assurance level has been chosen commensurate with the threat environment that is experienced by typical consumers of the TOE. In this Security Target the physical protection is left to the TOE environment, i. e. the TOE is expected to be operated in some secure environment. The protection of the TOE by the physical enclosure and sensors of the HSM is currently out of the scope of this software-only evaluation. A re-evaluation including the physical protection features will be considered once the new hardware is available.

6.3 Security Functional Requirements Rationale

6.3.1 Dependencies

All dependencies mentioned in CC Part 2 for the SFRs have been met, except the following:

- **FCS_CKM.4**, as formally required by **FCS_COP.1/SHA** has been omitted, because hash functions do not use keys that would require destruction. Any claim for key generation is not applicable, either.
- **FTP_ITC.2** dependencies to either **FTP_ITC.1** or **FTP_TRP.1** have been omitted. Communications between multiple TOE instances are indirect, are authorized by Administrators, and do not implement a single trusted channel. (Indirect communications are logically equivalent to multiple, independent administrative/management activities **FMT_MTD.1/Admin**.)

6.3.2 Coverage of Security Objectives

The security objective **O.Endorsed_Crypto** (Endorsed cryptographic functions) requires the TOE to provide Endorsed cryptographic functions and Endorsed cryptographic protocols to protect the user data as required by **OSP.User_Data_Prot** and for key management. This security objective is provided by the SFR: **FCS_CKM.1/AES**, **FCS_CKM.1/TDES**, **FCS_CKM.1/RSA**, **FCS_CKM.1/DSA**, **FCS_CKM.1/EC**, **FCS_CKM.2/Import**, **FCS_CKM.2/Export**, **FCS_CKM.4**. Also the SFRs **FCS_COP.1/AES**, **FCS_COP.1/TDES**, **FCS_COP.1/RSA**, **FCS_COP.1/DSA**, **FCS_COP.1/ECDSA**, **FCS_COP.1/ECDH**, **FCS_COP.1/SHA**, and **FCS_RNG.1**, which require meeting Endorsed standards for cryptographic functions. **FDP_ITC.2** and **FDP_ETC.2** enforce the use of Endorsed cryptographic functions for import and export of confidential cryptographic keys.

Hash functions supported by the TOE, while not depending on keys, similarly meet Endorsed standards for cryptographic functions (**FCS_COP.1/SHA**).

The security objective **O.I&A** (Identification and authentication of users) requires the TOE to identify uniquely users and to verify the claimed identity of the user before providing access to any controlled resources with the exception of read access to public objects. This security objective is provided by the following SFR:

- **FIA_UID.1** allows unidentified users to execute TOE operations that are neither restricted to an administrator, nor use session bound objects, nor are restricted in general by the control point access control policy of the TOE only.

  Conversely, **FIA_UID.1** requires identification before any other TSF mediated action.
- **FIA_UAU.1** allows Unauthenticated users to execute TOE operations that are neither restricted to an administrator nor are restricted in general by the control point access control policy of the TOE. The TOE requires identification selection of a claimed role and requires authentication before any other TSF mediated action.
- **FIA_AFL.1** requires detection and reaction to unsuccessful authentication attempts.
- **FIA_ATD.1** requires maintaining security attributes to individual users including Identity, Role and Reference authentication data as prerequisite for identification and authentication of users.
- **FIA_USB.1/GE** and **FIA_USB.1/IU** associating the identity and the role with the subjects acting for the authenticated user.
- **FMT_MTD.1/Admin** restricts the creation, clearing and deletion of Authentication Reference Data to the role Administrator.
- **FMT_MTD.1/User** restricts the ability to modify the Reference authentication data the user to which belongs this security attribute.

The security objective **O.Roles** (Roles known to TOE) is implemented by the SFR **FMT_SMR.1** which requires the TOE to provide at least the Administrator, Unidentified user and Unauthenticated user roles.

The security objective **O.Control_Services** (Access control for services) requires the TOE to restrict the access to its services, depending on the user role, to those services explicitly assigned to the role. Assignment of services to roles shall be either done by explicit action of an Administrator or by default. This security objective is provided by the following SFR:

- **FDP_ACC.2/CP** and **FDP_ACF.1/CP** require access control to the general services of the TOE per domain,
- **FMT_SMF.1** lists the security management functions.
- **FMT_SMR.1** describing the minimum list of roles and restrictions to these roles.
- **FMT_MTD.1/CP** limits the management of domain control points to administrators.
- **FMT_MSA.1/Key_Man_1** and **FMT_MSA.1/Key_Man_2** require limitation to the management of security attributes of cryptographic keys, key components and CSP describing the available services for these objects.
- **FMT_MSA.2** and **FMT_MSA.3** describe additional requirements to the management of security attributes to enforce the access control SFP for **FDP_ACF.1/OSA** and **FDP_ACF.1/SO**.

The security objective **O.Control_Keys** (Access control for cryptographic keys) requires the TOE to restrict the access to the keys, key components and other CSP according to their security attributes. This security objective is provided by the following SFR:

- **FDP_ACC.2/OSA** and **FDP_ACF.1/OSA** require access control to the key keys, key components and other CSP according to their security attributes,
- **FDP_ACC.2/SO** and **FDP_ACF.1/SO** require access control to the keys and other CSP of the TOE according to the session they belong to.
- **FMT_MSA.1/Key_Man_1** and **FMT_MSA.1/Key_Man_2** require limitation to the management of security attributes of cryptographic keys, cryptographic key components and CSP describing the access rights, available services and properties for these objects.
- **FMT_MSA.2** ensures that only secure values for cryptographic keys, key components and CSP are accepted for security attributes.
- **FDP_IFF.1** requires enforcement of security attributes, including domain access control policies.
- **FDP_IFC.2** requires enforcement of domain access control policies on requests and operations involving subjects in the TOE.

The security objective **O.Audit** (Audit of the TOE) requires the TOE to provide the capability to create audit records of security relevant events associated with users. This security objective is provided by the following SFR:

- **FAU_GEN.1** lists the auditable events to be provided by the TOE,
- **FAU_SAR.1** requires to provide at least with Crypto Officer and Administrator the capability to read all audit data from the audit records
- **FAU_STG.1** requires protection of the stored audit records from unauthorised deletion and prevention of modification.
- **FAU_STG.4** shall prevent loss of audit data if the audit trail is full by overwriting the oldest stored audit records.
- **FPT_STM.1** requires the TOE to provide reliable time stamps for its own use.

The security objective **O.Key_Management** (Management of cryptographic keys) requires the TOE to manage securely cryptographic keys, cryptographic key components and CSP. This security objective is provided by the following SFR:

- **FCS_CKM.1/AES**, **FCS_CKM.1/TDES**, **FCS_CKM.1/RSA**, **FCS_CKM.1/DSA**, **FCS_CKM.1/EC** and the lifecycle-related **FCS_CKM.2/Import**, **FCS_CKM.2/Export**, **FCS_CKM.4** provide the Endorsed cryptographic functions used by key management.
- **FDP_ITC.2** and **FDP_ETC.2** ensure the import and export of cryptographic keys, cryptographic key components and CSP with security attribute, which are associated with these objects for key management.
- **FMT_SMF.1** list the security management functions and **FMT_SMR.1** the roles for key management (i.e. Administrators for administrative, Users for operational keys).
- **FMT_MSA.1/Key_Man_1**, **FMT_MSA.1/Key_Man_2 FMT_MSA.2** and **FMT_MSA.3** describes the management of security attributes of cryptographic keys, cryptographic key components and CSP.
- **FPT_TDC.1** ensures the consistency of the security attributes of cryptographic keys, cryptographic key components and CSP.

The security objective **O.Key_Export** (Export of cryptographic keys) requires the TOE to export keys with their security attributes and protected in integrity. This is provided by the following SFR:

- **FCS_CKM.2/Export** requires the TSF to distribute keys by export methods meeting Endorsed standards and provides a refinement for keys exported for manual import.
- **FDP_ETC.2** requires the TSF to export keys unambiguously associated with their security attributes.
- **FPT_TDC.1** requires to ensure inter-TSF basic TSF data consistency for exported security attributes of cryptographic keys, key components and CSP.
- **FCS_COP.1/Backup_Enc** requires the TSF to export state encrypted using Endorsed standards (including any key material included in exported state).
- **FCS_COP.1/Backup_Int** requires the TSF to export state with integrity protection.

The security objective **O.Key_Generation** (Generation of cryptographic keys by the TOE) requires the TOE to generate cryptographic strong keys using Endorsed cryptographic key generation algorithms. This is provided by the SFR **FCS_CKM.1/AES**, **FCS_CKM.1/TDES**, **FCS_CKM.1/RSA**, **FCS_CKM.1/DSA**, **FCS_CKM.1/EC**, which require the use of Endorsed key generation algorithms and **FCS_RNG.1** describing requirements for the random number generator needed for key generation.

The security objective **O.Key_Import** (Import of cryptographic keys) requires the TOE to import keys with security attributes and verify their integrity. The TOE shall import secret or private keys in encrypted form or manually using split knowledge procedures only. This is provided by the following SFR:

- **FCS_CKM.2/Import** requires the TSF to distribute by key import methods meeting Endorsed standards and provides a refinement for manually imported keys.
- **FDP_ITC.2** requires the TSF to import keys unambiguously associated with their security attributes.
- **FPT_TDC.1** requires to ensure inter-TSF basic TSF data consistency for imported security attributes of cryptographic keys, key components and CSP.
- **FPT_ITI.1** requires the TSF to detect modification of imported data, including keys, key components, and CSP.

The security objective **O.Key_Destruction** (Destruction of cryptographic keys) requires the TOE to destruct keys cryptographic key components and other CSP on demand of users or when they will not be used any more in a secure way that no information about these keys is left in the resources storing or handling these objects before destruction. This is provided by the following SFR:

- **FCS_CKM.4** requires the TSF to provide Endorsed mechanisms for key destruction.

The security objective **O.Check_Operation** requires the TOE to perform regular checks to verify that its components operate correctly including integrity checks of TOE software, firmware, internal TSF data and keys. This is provided by the SFR:

- **FPT_TST.1** requiring TSF self tests.
- **FPT_FLS.1** requires the TSF to preserve a secure state when self-test fails.

The security objective **O.Prevent_Inf_Leakage** (Prevent leakage of confidential information) requires the TOE to prevent information leakage about secret and private keys and confidential TSF data outside the cryptographic boundary and unintended output confidential user information. This is provided by the following SFR:

- **FDP_RIP.2** requires the TOE to ensure that any previous information content of a resource is made unavailable.
- **FCS_CKM.4** requires the TSF to provide Endorsed mechanisms for key destruction.

7 TOE Summary Specifications

The TOE implements the following set of security functions to address requirements:

Secure key management, internal. Generation and proper lifecycle-tracking of keys within the TOE

Secure maintenance of host-based keystores. Preserving the confidentiality and integrity of sensitive data, including keys and sessions, while exported from the TOE.

Security attributes of keys are maintained both in TOE-internal form, and when keys are transported through PKCS#11-derived `WrapKey` and `UnwrapKey` calls. Sessions and other non-key state only have TOE-internal forms: unlike keys, they are not transported over module PKCS#11 logical boundaries.

Cryptographic operations. Services provided by the TOE, both for internal and host-application use (Table 4).

Application identification and authentication. Unambiguous identification of application identities for keys and sessions bound to specific identities (jobs, processes, etc.).

Policy enforcement. Reliable enforcement of usage restrictions based on session, object, TOE policy, or their combination.

Usage restrictions, representation and enforcement. Hierarchical enforcement of different kinds of restrictions, implementing the infrastructure which may be used to enforce policies.

Administrative services. Identity-based access to security-relevant management functions and internally stored administrative data.

Selftests. Compliance and integrity tests of cryptographic primitives for functionality usable by the TOE.

Audit. Reporting security-relevant events for logging. Query capabilities for auditing security-relevant data within the TOE.

Random-number generation. Generate a bitstream entirely within the TOE, suitable for cryptographic purposes such as key generation.

Additional functionality, other capabilities not included in top-level security functions listed above:

Key virtualization. Segmentation of keystores based on trust root, application identity, or the combination of these. Management of multiple trust roots.

Secure storage. Integrity protection and owner identification for keys, sessions, or other security-critical date when inside the TOE. This category includes both long-term resident data and transient, per-request objects.

7.1 Security Functions

Please note that some functions described in the following sections may not be part of the TOEs evaluated security functionality. Only functions described in the SFRs of section 6.1 have been subject to this evaluation. Other functions have been described here for a better understanding of the whole product and completeness reasons only.

7.1.1 Secure key management, internal

Keys generated by, or imported to the TOE are associated with their attributes, and are not separated in the evaluated configuration. Internally generated and imported keys differ in their PKCS#11-standard `CKA_LOCAL` attribute—which indicates TOE-internal generation, when true—but are managed identically otherwise. The `CKA_LOCAL` attribute is supplied by the TOE, it is maintained together with key material, and it is never modifiable (SFR FDP_ACF.1)

The TOE supports the PKCS#11-standard `CKA_MODIFIABLE` attribute to separate read-only and modifiable keys. Following PKCS#11-prescribed behaviour, transition to non-modifiable state — removing `CKA_MODIFIABLE` — is non-reversible. An additional attribute, `CKA_NEVER_MODIFIABLE`, indicates the history of the key (whether it has been created as read-only). The attribute is maintained together with key material, and it is never modifiable (SFR FDP_ACF.1)

A non-standard, weaker form, `CKA_IBM_RESTRICTABLE`, is provided to mark keys which may accommodate further capability-restricting modifications, but are not eligible for capability-extending changes. This extension allows the TOE to derive multiple forms of the same key without making them all read-only, but still preventing the derived keys from activating attributes not allowed by the originator. As an example, one may derive a MAC-Verify key from a base symmetric key (lacking the capability to Sign), knowing that the owner of the Verify-only key is incapable of re-activating the `CKA_Sign` attribute even if allowed to restrict the key.

The TOE uses the PKCS#11-standard `CKA_EXTRACTABLE` attribute to prevent key export. The corresponding, read-only `CKA_NEVER_EXTRACTABLE` stores past history, indicating when a key has been created so.

In the PKCS#11 meaning, all TOE-managed keys are "sensitive", and imply the `CKA_SENSITIVE` attribute. At least one host PKCS#11 provider uses this attribute to select when a "protected-key" PKCS#11 object needs to be routed to Enterprise PKCS#11.

Lack of key expiration Note that the stateless TOE lacks the concept of key expiration. If desired, keys may be bound to time-restricting sessions, forcing expiration of all associated objects upon session removal (`Logout`). Similarly, all objects encrypted by a specific WK are invalidated if the controlling WK is ever changed, even if the actual objects reside by users, disjoint from Administrators controlling WKs.

All transient administrative keys — such as import-controlling public keys — are destroyed upon the first successful use, and require no further temporal restrictions (SFR FDP_RIP.2). These keys are only saved within transient memory internally, and they must be newly generated if the HSM containing them gets restarted before their use.

The TOE, as an implementation of the standard PKCS#11 API, acknowledges the possibility of key expiration, but makes it clear that temporal restrictions are outside its scope, therefore it ignores time-based key expiration. This matches the security rationale of PKCS#11 itself [PKC04, 10.6.2,10.7.2], and it is mentioned here for completeness.

7.1.2 Secure maintenance of host-based keystores

Sensitive data including keys and session state, when exported from the TOE, reside on untrusted hosts. The TOE protects both the integrity and confidentiality of any such sensitive data, and unambiguously associates it with any controlling entities. Sensitive data does not leave the TOE in cleartext.

Host-resident sensitive data is protected by authenticated encryption, with exactly one of multiple possible wrapping keys — WKs — and associated MAC keys, maintained within the TOE. The authenticated encryption is a standard Encrypt-then-MAC construct, using symmetric encryption with HMAC-based integrity protection [Kra01, BN08] [Sma13, 4.3.1]. Host-resident state carries its internal attributes; see section 8.3.2 for a list.

All sensitive data is unambiguously associated with a controlling WK, and optionally with an identity-derived *session identifier,* if the controlling application requests objects bound to sessions. (SFR FIA_ATD.1, SFR FIA_USB.1/GE, SFR FDP_ACC.2/SO) The TOE creates token-scoped "token objects", not bound to any specific authenticated user [PKC04, 6.4] when no session identifier is used (SFR FIA_USB.1/GE, SFR FIA_UID.1 and SFR FDP_ACF.1/SO).

The session identifier is constructed through a call to the `Login()` service (SFR FIA_USB.1/GE); it serves as a Boolean pass/fail indicator of presence/absence of a session-associated entity, but contributes no additional attributes. The validity of the session is, however, evaluated before checking any of the object-embedded attributes.

Internally, multiple WKs are stored and selected based on infrastructure-level information, which are transparent at the PKCS#11 service level. Each internal segment — "domain" — uses a dedicated WK for its own sensitive data (SFR FIA_USB.1/GE). Objects belonging to different WKs are mutually incompatible. Rollover of a WK, or its destruction prevents use of key material or state bound to that WK (SFR FDP_ACF.1/SO). See "Key virtualization" (7.1.11) for the details of WK separation.

Certain non-sensitive objects, specifically public keys, may also be bound to specific WKs. Such objects are used, for example, where the TOE must associate non-sensitive data with attributes. These non-sensitive structures are authenticated by an HMAC, sharing a MAC key with their sensitive — private-key — counterparts. These non-sensitive objects also include attributes.

Transport of security attributes In the evaluated configuration, functional key transport is possible only with "attribute-binding," a non-PKCS#11 key transport method where keys and attributes are not separated from each other. The corresponding format unambiguously specifies attributes within the encrypted — and authenticated — wrapped key object. *These security requirements rule out all standard-defined PKCS#11 key-transport modes, necessitating the definition of our proprietary — but documented — formats.* (SFR FDP_ETC.2, SFR FDP_ACF.1/OSA)

Host-resident keys, both encrypted blobs and authenticated public keys, always include their security attributes; see section 8.3.2 for an attribute listing. (SFR FDP_ETC.2, SFR FDP_ITC.2, SFR FDP_ACF.1/OSA)

Administrative key transport relies on implicit interpretation of keys — also authenticated by regular administrative procedures — and therefore does not explicitly support attributes.

Both administrative and non-administrative key transport, in the TOE configuration, supports authenticated-encrypted formats, and rejects transported keys when modifications are detected. (SFR FPT_ITI.1) Data inconsistencies, when encountered, are interpreted as corrupted data; note that formats are versioned to allow version-dependent migration (and therefore the TOE may be deployed without the need to interpret unknown structure versions) (SFR FPT_TDC.1, SFR FDP_ITC.2)

Administrative keys are identified through whitelists. While X.509 certificates are used to identify *public keys,* no additional attribute is interpreted. Note that presence on the whitelist is sufficient information for authorization, therefore attributes within X.509 certificates [RHPFS02, 4.2] are ignored. The lack of attribute interpretation by the TOE immunizes it to attacks on certificate (trust) chaining [BJR+14, IX].

7.1.3 Cryptographic operations

The TOE provides cryptographic signing/verification, key generation and derivation, hashing, encryption and decryption services to the host it is attached to, or remote clients of that host. The external API provided by the TOE is derived from PKCS#11 [PKC04], with adaptation to accommodate a mainly-stateless instantiation.

The following groups of functional services are offered to users:

1. Generate or derive keys: AES, TDES, RSA, EC (elliptic curve, prime field, NIST P-curves or BP ones), DSA, DH, generic secret keys (a PKCS#11 abstraction for arbitrary, unstructured sensitive data)

   (SFR FCS_CKM.1/AES, SFR FCS_CKM.1/TDES, SFR FCS_CKM.1/RSA, SFR FCS_CKM.1/DSA and SFR FCS_CKM.1/EC)

Algorithm	*Key size (bits)*	*Mode*	*Operation*
Symmetric algorithms			
AES	128, 192, 256	ECB,CBC	encryption, decryption
TDES	192 (168 effective)	ECB,CBC	encryption, decryption
Asymmetric algorithms			
RSA	2048, 3072, 4096	N/A	signing, signature verification, encryption, decryption
EC (ECDSA, ECDH)	NIST/BP sizes	N/A	signing, signature verification, key agreement (NIST or Brainpool prime-field curves) P-192, P-224, P-256, P-384, P-521, BP-192, BP-224, BP-256, BP-320, BP-384, BP-512 [R, twisted]
DSA	2048, 3072	N/A	signing, signature verification
Hash and MAC algorithms			
SHA-1, 224, 256, 384, 512, 512/224, 512/256	N/A	N/A	(hashing)
Deterministic random-number generator (RNG)			
ISO 18031 (Hash-DRBG, SHA-256)	N/A	N/A	

Table 4: Algorithms supported by the TOE

2. Generate or verify digital signatures with asymmetric keys [Nat13a]
   (SFR FCS_COP.1/RSA, SFR FCS_COP.1/ECDSA, and SFR FCS_COP.1/DSA)
3. Key agreement (SFR FCS_COP.1/ECDH)
4. Encrypt or decrypt data with asymmetric keys (SFR FCS_COP.1/RSA)
5. Encrypt or decrypt data with symmetric keys (SFR FCS_COP.1/AES, SFR FCS_COP.1/TDES,
6. Cryptographic hash functions (not requiring WKs) (SFR FCS_COP.1/SHA)
7. Random-number generation (not requiring WKs) (SFR FCS_RNG.1)
8. Storage, use, and disposal of secrets within the TOE
9. Query TOE-internal capabilities

Services are available to anonymous users, without identification, if the host grants access to the TOE. *Host-based traffic restrictions may be implemented, but they are not visible to the TOE, and are outside the scope of this security target.*

Services involving sensitive state require an installed, active WK in the domain they are requested from. The small subset of services which do not involve sensitive data — hashing and random-number generation — may be serviced even by domains lacking active WKs.

Administrative and other services built on top of cryptographic primitives are used by the TOE. They are discussed in their respective sections.

Queries returning information about TOE capabilities are not access-controlled (SFR FIA_UAU.1).

7.1.4 Application identification and authentication

The TOE distinguishes applications using its services based on either proof of possession, or cryptographic authentication, in the cases where users of services need to be identified. Beyond these identification methods, the TOE lacks the concept of actual users. (SFR FMT_SMR.1)

Non-administrative services are available without authentication, unless they reference host-resident state which involves *sessions*. (SFR FIA_UID.1). Objects bound to sessions are usable as long as the corresponding session is registered within the TOE. Sessions are ordinarily registered or removed through the `Login` and `Logout` services, which are available to all users eligible to support requests (SFR FIA_USB.1/IU, SFR FDP_ACF.1/SO, SFR FMT_MTD.1/User_1, SFR FMT_MTD.1/User_2, SFR FIA_UAU.1)

Session identifiers are derived from host-supplied data, and are therefore influenced, but not fully determined, by host-visible information, such as passphrases. Host code is responsible for separating multiple entities — jobs, processes — as cryptographic identification is not available at this stage. The session derivation process calculates an HMAC incorporating

Functionality	*Key type and size (bits)*	*Mode*	*Operation*
Functional keys controlling PKCS#11 key objects			
Wrapping key (per domain)	AES/256	encryption (keys or state)	Encrypt host-resident objects. One instance per domain. Possibly diversified through session identifiers.
Domain MAC key	HMAC-SHA256/256	generate/verify object MAC	Integrity-protection of host-resident objects. Derived inside TOE from corresponding WK.
Session-unique keys (transient)	AES/256	encryption (keys or state)	Encrypt host-resident objects, if bound to sessions. Derived as transient key, disposed immediately after use.
Administrative keys, persistent			
Administrator identity	RSA or EC	verify	Verify: signatures on authenticated commands. Only public key resides within TOE.
Device keypair	RSA/4096	sign	Sign: module-originated responses. Generated inside the TOE, never exported.
Device certificate	RSA/4096	verify	of device keypair: exported to allow device attestation.
Administrative keys, transient			
Transport key, symmetric	AES/256	encrypt or decrypt	encrypts serialized administrative state. Unique key used by each export. Destroyed after export.
Key-exporter, KEK	RSA or EC	encrypt	Asymmetric-encryption transport key or its keyparts. Only public key is resident within TOE: imported by administrators with export request. Private key resides with "keypart holders", outside TOE.
Key-importer, KEK	RSA or EC	decrypt	Asymmetric-decrypt transport key or its keyparts. Destroyed after successful import. Never exported.
Functional keys for PKCS#11 use			
Any PKCS#11 key	as requested by PKCS#11 applications	any supported	Used together with a controlling WK, optionally with an additional session identifier.

Table 5: Key types maintained within the TOE

host-supplied information, and is assumed to be one-way, not revealing the originating host data. *Once generated, the TOE assumes that host entities possessing the session identifier are authorized to access any host-resident state bound to that session — i.e., session-users authenticate through proof of possession* (SFR FIA_USB.1/IU)

Loss of the derived session identifier — a MAC — is expected to be insufficient to recover the originating identification data [Mur08, 3]. Once created, session identifiers are immutable (SFR FIA_USB.1/IU).

Once a session has been established, parts of its identifier is embedded into all objects bound to that session (SFR FIA_USB.1/GE). The partial identifier is host-visible, but is insufficient to recreate the full session identifier. Requests using a session-bound object are rejected when the session is unregistered from the TOE (through `Logout()`). Removing a session without administrator intervention requires proof of possession: only those possessing the full session identifier may unregister the session. (SFR FDP_ACF.1/SO)

As a special case, a signed administrative command is eligible to remove any session from the TOE. This possibility exists to forcibly remove sessions if the originating passphrase may not be recreated; it is beyond the capabilities of any PKCS#11 entity.

Administrators are identified through their X.509 public-key certificates, which are loaded into the TOE, with the corresponding private keys resident outside the TOE. Administrators are identified through signatures on state-changing administrative commands:

- Lists of eligible administrative certificate is maintained inside the TOE at all times. Management of the lists is a regular signed administrative service.
- Signed commands include industry-standard `SignerInfo` structures, specifying signer identity — i.e., SKI, *Subject Key Identifier*, hash of the entire public key — and signature algorithms
- Signed commands are unambiguously targeted to the whole card or a particular domain, including card serial number (which the TOE uses as provided by infrastructure, but is unable to change)
- Transaction counters are maintained to enforce freshness: each signed command must advance a monotonically increasing counter within the TOE
- random nonce-like "instance identifiers" are generated upon each zeroization — return to factory state — to prevent replay of previous administrative traffic

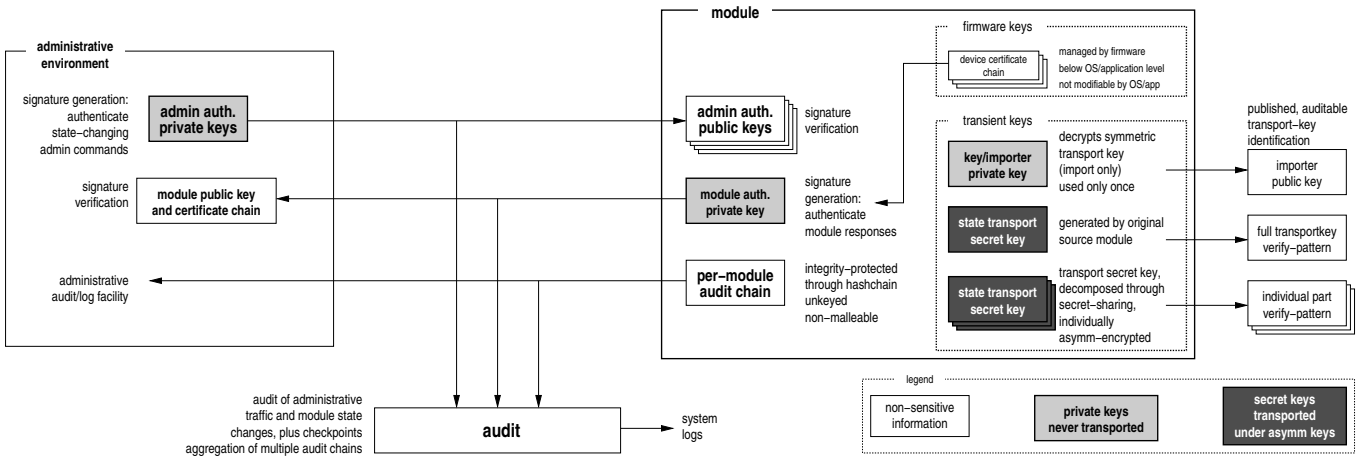

Figure 3: Functionality and location of administration-relevant keys

Note that administrative certificates use X.509 structures as portable containers for public keys. Any attributes within certificates [RHPFS02, 4.2] are ignored, since we only infer identification from administrator whitelists. The lack of attribute interpretation by the TOE immunizes it to attacks on certificate (trust) chaining [BJR+14, IX] [GIJ+12, 7].

Lists of administrators are maintained for the whole card, and separate lists exist for each internal domain. When identifying a signature, a sufficient number of disjoint administrators must sign the command, with the accepted list depending on the targeted entity (see Fig. 3):

- Card-level commands must be signed by card-level administrators. These administrators have the highest possible precedence, since they may administer domain administrators.
- Domain-level commands may be signed either by administrators of the targeted domain, or module administrators. We recommend keeping the sets of administrators disjoint, and not mixing administrators as normal practice.

All administrator-signature verification steps are unambiguous: the set of eligible administrators is known, and the required number of signatures is determined from the command and the attribute setup of the targeted entity.

7.1.5 Policy enforcement

Usage restrictions combine object-level attributes and those of the domain where the request is executed. Objects feature fundamental functional restrictions, such as allowing encryption/signature generation/etc. by a given object, with the following capabilities stored within each object:

1. encrypt or decrypt data
2. sign data or verify signatures on data
3. wrap or unwrap other keys
4. derive keys from the current one, including some key agreement algorithms (which are grouped under key derivation, not un/wrapping, under PKCS#11)
5. object attributes may be modified
6. attributes may be removed from the object, but not added (custom attribute)
7. object is Attribute-Bound, never transported without attributes (custom additions to PKCS#11)

Most of the above capabilities correspond directly to standard PKCS#11 Booleans. Interpretation is polymorphic: decryption for RSA keys affects a different set of algorithms than decryption for symmetric keys, but these capabilities are orthogonal to algorithms or keytypes. Similarly, sign or verify are interpreted in the context of key type: asymmetric keys' private objects may only sign, while their public counterparts may only verify. (SFR FDP_ACC.2/OSA, SFR FDP_ACF.1/OSA)

Attributes are supplied with the object during key generation, and are stored authenticated within objects; appropriate defaults are provided if they are not supplied with the request (SFR FMT_MSA.3). Boolean attributes and expected-compliance profiles are replicated to the clear object header, so policy compliance may be audited outside the TOE. The TOE rejects clearly invalid combinations of attributes, if conflicting ones are supplied, or if requested attributes are rejected by policy, such as when considered insecure (SFR FMT_MSA.2). (Note that the interpretation of "insecure" is dynamic, depending on CP setup.)

Certain attributes are specific to EP11:

1. object attributes may be removed, but not added, if the `RESTRICTABLE` attribute is present
2. object may be marked as *attribute-bound,* not eligible for transport where attributes and key material may be separated

See 8.3.2 for a full listing.

Each domain features a static list of *control points,* which generally restrict operations (8.3). Control points form a single linear list, assigned without hierarchy, and are a mixture of generic and very specific restrictions. The TOE does not provide tools for grouped management of control points, or map arbitrary abstract policies to collections of control points: these are assumed to be undertaken on the host. Compliance with the most relevant security profiles, as derived from the control point setup, is passively reported (see below). (SFR FDP_ACC.1/CP, SFR FDP_ACF.1/CP)

Attribute enforcement follows a conservative default: operations MUST be actively enabled by attributes, control points etc. Conversely, none of the requested functions may be prohibited by any active attribute. (SFR FDP_ACF.1/OSA)

For an object to be operational, the domain hosting the request must allow the operation (SFR FDP_IFF.1), the object must not be restricted from executing that operation, and the corresponding attributes and algorithms ("mechanisms" in PKCS#11 terminology) must be consistent with the intended use. (SFR FDP_IFC.2, SFR FDP_IFF.1)

The TOE also enforces *standards-compliance profiles,* indicating when the active set of control points prescribes an assumptions compatible with specific — revisions of — security standards. Currently, the following compliance profiles are reported:

1. compliance with FIPS 140-2 restrictions, algorithms and minimum key sizes. Two versions, differentiating FIPS 140-2 in 2009 and 2011. (Advancing to compliance with the 2011 profile prohibits algorithms with 80-bit strength, and some minor algorithmic restrictions.)
2. compliance with key-related functional requirements derived from [BSI08] requirements (excluding hardware security), and algorithm selections corresponding to the then-current rules of the Bundesnetzagentur. Two versions, for years 2009 and 2011, representing changes in algorithmic restrictions.

Compliance profiles are reported as a passive entity: one may set control points, and compliance profile is derived from that set. The host may support profiles, and turn them into filter conditions for control points, which is outside the TOE.

Compliance profiles are controlled at a domain level, but they are also reported as aggregates. The card reports the compliance profile as a logical AND of all active domains' compliances. Therefore, if the card compliance profile reports compliance bit N, all active domains will be known to comply with standard(revision) N.

Reporting of all control points or compliance profiles is available as a signed, audit-ready administrative query

7.1.6 Usage restrictions, representation and enforcement

Restrictions on the use of attribute-equipped blobs are enforced in a hierarchical set of checks. Allowing or rejecting use of any object is controlled by the combination of all restriction categories (Fig. 4), allowing only the subset of objects allowed by all:

1. *Control Points* (CPs) of the responsible domain, representing a diverse set of functional-level usage restrictions (see sections 8.3 and 8.3.1 from page 45). (SFR FDP_ACF.1/CP)

   While CPs restricting keytype/strength/mode form clearly defined groups, most CPs are not so categorized. These "other" CPs impose usage restrictions on very diverse points of backend control flow, therefore the highlight in Fig. 4 with no direct connection.

2. Administrative attributes derived from per-domain CP setup, or their card-level aggregated equivalents. Fig. 4 shows this relationship, when domain/card compliance attributes are derived from CPs.

   Compliance attributes, as an example, are a condensed representation of multiple CPs. They are provided as read-only attributes to show compliance — or lack thereof — with security standards or regulations.

3. Key size, type, initialization state, usage restrictions of the blob-internal key/state, as recovered from the blob.

   Blob attributes MAY be modifiable or amenable to subsequent restrictions (SFR FMT_MSA.1/Key_Man_2), or may be always non-modifiable (SFR FMT_MSA.1/Key_Man_1). Partitioning into non/modifiable is static: non-Boolean attribute types (such as key type, size, curve parameters etc.) are non-modifiable, Booleans MAY be modifiable. Booleans MAY be further restricted and non-modifiable, such as PKCS#11 EXTRACTABLE or MODIFIABLE attributes.

   Supporting modifiable, only restrictable, or non-modifiable objects allows us to implement specific policies derived from PKCS#11 and extended attributes (SFR FMT_MSA.3).

4. The list of sessions maintained adapted forms of PKCS#11 commands (7.1.4)

5. Restrictions on key size, type, initialization state, operational mode etc. imposed by the PKCS#11 API. These restrictions are encoded in control flow, are not runtime-controlled, and are inherited from [PKC04], adapted to our backend.

   API-imposed restrictions specify tuples such as: `Encrypt()` requires an encrypt/state object output by `EncryptInit()`, which has not yet used by an incremental `EncryptUpdate()`. The base key initializing encryption state must have been an encryption-capable keytype, and had its `CKA_ENCRYPT` attribute set. Data passed to `Encrypt()` may have size/format restrictions based on algorithm/mode etc.

As shown in Fig. 4, the different types of restrictions force decisions based on different properties of each object (SFR FDP_IFC.2):

- the availability of PKCS#11 services may be prohibited by CPs. As an example, use of the `WrapKey()` service is prohibited if CPs of the responsible domain prohibit any kind of key export, even before checking whether the supplied blob is `WRAP`-capable
- services generally accept only a subset of objects, which must be of the given mode; see `Encrypt()` example above.

  Object checking combines base functionality (`Encrypt()` requiring state initialized for encryption), past history (incremental and single-call `Encrypt()` calls may not be mixed), and other similar calls.
- both keytype and cryptographic strength of the object must be permitted by the current CP setup.

  Key type checks implicitly include algorithm-category checking: use of any kind of private key is separately controlled from use of elliptic-curve keys (or even use of specific categories of elliptic curves, see the defined CPs).
- the entire set of object attributes must be consistent with the CP set of the responsible domain. As an example, if the CP setup prohibits use of keys capable of both en/decryption and un/wrapping, en/decryption and un/wrap-related object attributes are cross-checked (such as: reject attacks mixing keys and data through encrypt+unwrap-capable dual-use keys [Clu03, 4]).

  Since these interactions involve entire sets of attributes, they are not separately shown in Fig. 4.
- if an object is session-bound, its controlling session must be active (present) in the targeted backend

  The PKCS#11 notion of a "private object" — those bound to logged-in PKCS#11 sessions — map to EP11 sessions, therefore we show a connection between sessions and PKCS#11-derived restrictions.
- for functional use, the compliance mode of the object must match the then-current administrative compliance attribute. (This, in turn, is a condensed representation of the full set of CPs.)

Objects must be allowed by each of these restrictions to become eligible for use. Failures are reported as a combination of standard PKCS#11 errors, such as *CKR_KEY_TYPE_INVALID*, and some IBM-extended ones, mainly for CP-mandated errors (which have no PKCS#11 equivalent).

To somewhat simplify policy/setup-induced error reporting, we distinguish between policy rejections which *may* be disabled by CP changes, and failures where the responding backend can not be configured to accept that object. As an example, a future backend removing support for deprecated algorithms would reject existing blobs of this deprecated types with the latter policy rejection error.

Most of restrictions-enforcement is centralized: the backend uses a single `unwrap_blob()` service, which is responsible for all context-independent checking. Since most restrictions may change at runtime, all checking is applied against the then-current setup. *As with Unix file-permission checking, once the services requested by a call are approved, it may be allowed to complete, even if the setup changes after the check, during execution* (which, under regular operations, does not generally happen due to logistics/policy reasons).

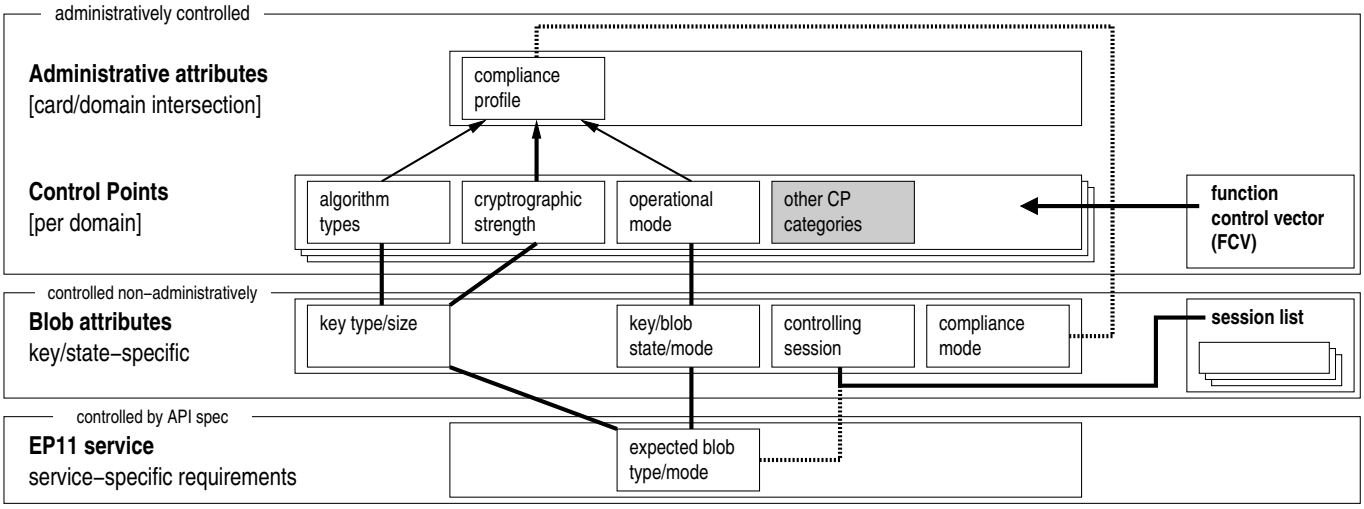

Figure 4: Usage restrictions: hierarchical enforcement

The specific order of checking restrictions is not specified. We generally minimize unnecessary computation, therefore restrictions which may be evaluated on plaintext-visible information — such as mechanisms — are applied before those dependent on blob-plaintext (such as blob-internal stream state). *We intentionally do not specify the specific order of evaluation, since that may need to change when restrictions are added in the future.*

We note here that a significant complexity of our regression suite is present only to ensure that specific usage-restriction errors are encountered where expected. While constructing error cases is not very complicated, ensuring that CP and other setup lets those invalid requests through to the targeted check, without triggering other errors, is quite complex. Recognizing the futility of completing this manually, we constructed many of these compound conditions based on conditionals derived by static analysis tools — note that BEAM, the IBM-developed static analyzer constructs compound statements immediately suitable for turning into such erroneous conditions [Bra00, 7] [BBS06, 6].

7.1.7 Administrative services

Administrative services include "passive" queries and commands; the latter change state. Administrative queries, as their non-administrative counterparts, are available to everyone capable of submitting requests.

Administrative commands belong to one of the following groups (SFR FMT_SMF.1):

1. Administrator management of card or domain administrators' public keys (SFR FMT_MTD.1/Admin)
2. Key or state import/export (multiple, related commands, see Fig. 5). These key/state transport operations are authenticated and change state.

   Administratively managed import/export of state may include keys in encrypted form, their associated transport keys, and other attributes (SFR FCS_CKM.2/Import, SFR FCS_CKM.2/Export).

   State import/export uses an authenticated-encrypted enclosure for combining encrypted and clear regions, serialized as a single non-modifiable unit (SFR FCS_COP.1/Backup_Enc, SFR FCS_COP.1/Backup_Int).
3. Generation and removal of importer private keys
4. Management of administrative attributes (non-functional restrictions), including signing thresholds
5. Management of control points (functional restrictions, for PKCS#11-equivalent services)
6. Zeroization of domains or card SFR FCS_CKM.4
7. Infrastructure management (real-time clock) (SFR FPT_STM.1)

As shown previously under authentication, administrators are identified through public keys, and signatures of a sufficient number of administrators. Signature thresholds are an actively managed attribute, and require administrator co-signatures to change. Lacking the required number of valid signatures—over the same request structure—forces rejection of administrative requests (SFR FIA_AFL.1). (We do not distinguish between the amount of rejected signatures when rejecting a request; each request and its signatures are considered in isolation SFR FIA_AFL.1).

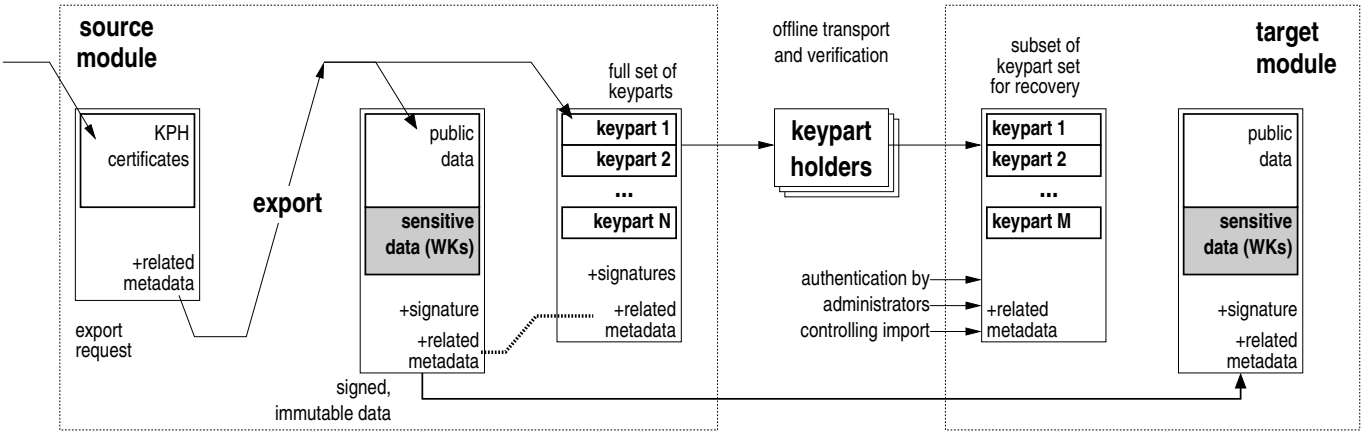

Figure 5: Authentication and data flow during backup/restore operations, including backend migration

Administrators are grouped into two hierarchical levels, card-level administrators managing shared infrastructure — such as the single clock — and domain administrators, and domain administrators managing domain settings.

Card-level administrators may supply signatures for domain commands, but the converse is not true.

Administrative commands changing state, with two exceptions, must be signed, with either a single signature, or a threshold number of signatures. (The choice of limit depends on command type.) Some of the services have a switchable threshold, and a single signature may suffice for an otherwise threshold-signed command. Unsigned administrative commands are provided for two commit operations which may be performed without explicit administrative operations; host infrastructure is allowed to issue them, and therefore they do not require authentication.

7.1.8 Selftests

During startup, or upon demand, a set of known-answer tests are executed. Failure prohibits subsequent cryptographic operations, and may be remedied by restarting the module. If the failure persists, the host — typically, driver infrastructure — must manage replacement or retries, and the TOE is no longer involved. (SFR FPT_TST.1, SFR FPT_FLS.1)

7.1.9 Audit

Security-relevant operations within the TOE are audited through an HSM-resident audit subsystem, based on a hash chain. Audit records form a logical sequence immune to insertion or deletion, inheriting the non-malleability of hash chains [SK99, 3] [Wou12, 6.2.4] [MA14, 5.4]. This property allows us to make statements about audit-chain security, even if the audit chain itself is eventually maintained outside the TOE (SFR FAU_STG.1). (The module retains only the most recent entries, and provides context fields such as sequence numbers, for their unambiguous identification.)

Audit records may contain fields which are public, and indirect, non-sensitive information derived from keys, but NEVER sensitive values. Indirect information, such as types, sizes, or truncated hashes of keys, MAY be logged. This is not a simple policy-driven restriction: the audit-event format simply lacks fields to contain/log sensitive data. (The only non-public key parameter logged, key checksum, follows PKCS#11 key-checksum rules, revealing a 24-bit truncated hash based on sensitive values.)

Audit records, when queried through non-administrative query, are returned without additional signatures. Administrative query responses are returned digitally signed, signing the same audit-record content, when requested. *The signed, administrative audit-query is added to provide digitally signed, individually verifiable entries of the same hash chain.* Both queries are available to any host entity (SFR FAU_SAR.1); host environments, outside the scope of the TOE, SHOULD supply tooling to parse and manage audit records (SFR FAU_SAR.1).

Audit events are output from the TOE for external storage. Since administrative responses include then-current administrative state, including transaction counters, all signed audit events may be unambiguously ordered and verified, and interpreted even in isolation. (Other than adding digital signatures, the administrative query signs the same event records. *The signed, administrative audit-query is added to provide digitally signed, individually verifiable entries* of the same hash chain.)

The audit scheme is designed to provide full accountability of TOE-internal operations which may be verified by external parties ([MA14, 3] "Retention and disclosure"), and provide all necessary context to interpret logs even without access to the originating module. Since the necessary context includes both time and sequence number, logical reconstruction of audit

streams does not require the cooperation of any other entity [MA14, 5.1]. With module-specific identification, multiple audit streams aggregated to a single compound are expected to be easily processable even in a large system aggregating billions of entries — practically within reach of even modest desktop systems as of this writing [SMZ14, 3.3].

The following events — among others — all generate audit records (SFR FAU_GEN.1):

1. Startup and shutdown of the TOE, and that of relevant subsystems — including audit functionality itself
2. Original and updated time of the TOE, when the corresponding administrative command updates the — module-global — time
3. Completion of selftests
4. Import of keys or key components (see key-audit records below) (a state-changing action)
5. Export of keys or key components (a state-changing action)
6. Destruction of HSM-resident keys (a state-changing action)
   Note that the stateless nature of PKCS#11 makes it impossible to audit destruction of host-resident objects.
7. Startup and recovery from detected failures of TSF (SFR FPT_FLS.1)

Among other attributes, each audit record includes at least the following metadata:

1. Sequence number of the audit event (SFR FAU_GEN.1)
2. Audit-instance identifier of the event log, allowing disambiguation even if the hosting HSM is zeroized
3. Time of the audit event, as reported by the HSM system clock
4. Type of event
5. Identity of the triggering session — if applicable — or the relevant administrator public key [in truncated form]
6. Identity — such as PKCS#11 checksum or public-key hash — of the key or keys involved in an event-relevant operation
7. HSM-generated salt of at least 96 bits, to prevent the host from advancing the audit hash chain in a fully host-controlled manner

Since software updates are prohibited by the current security target, audit capabilities related to updates do not apply to — are not expected to be encountered during operation of — the TOE.

To reliably frame the time reference around TOE time changes, two audit events are generated when the administrative command is issued. Both original and updated time are logged. Note that since the audit sequence number is not changed during time changes, the audit chain may be reliably reconstructed even in such cases. Sequence number tracking and forward integrity provided by the audit hashchain allows interested entities to establish "known good" synchronization points in each audit chain. Similar checkpointing is routinely used by large hashchain-based, publicly verifiable integrity proofs to allow long hashchain history [dcd14].

Since the audit subsystem is only observed, but not explicitly managed or controlled by any host entity, there are no additional TOE requirements for their protection. Therefore, security management of audit storage is not applicable to the TOE. Since the audit flow is based on hash chains, even untrusted hosts may manage audit records in a trustworthy and secure manner, assuming modifications may be detected by any other "auditor" (SFR FAU_STG.4). This approach assumes that anyone may query audit contents, an approach known to work for management of trust root certificates [LLK13, 3].

Non-administrative audit queries return raw audit events, without card-generated digital signatures. While these audit events are not signed, they are aggregated through a non-malleable hash chain. The TOE manages a single hash chain internally, and relies on the host reassembling hash chains, aggregating chains for multiple backends, and providing audit facilities for the entire audit trail.

Since the TOE relies on the host storing keys and sensitive state during regular operations, one may not meaningfully audit object destruction. Therefore, audit events related to non-administrative object removal are limited to TOE-internal objects, such as retained keys.

Data structures of the audit subsystem are not visible to the host, and no service provides access to them. If the core data structure is ever corrupted, a new audit chain is started, logging the time when this condition has been encountered 8.3.3. Initial startup, when a new audit chain is started without finding a previous structure is not separately reported, although the initial state will be obvious (sequence number starts at 0).

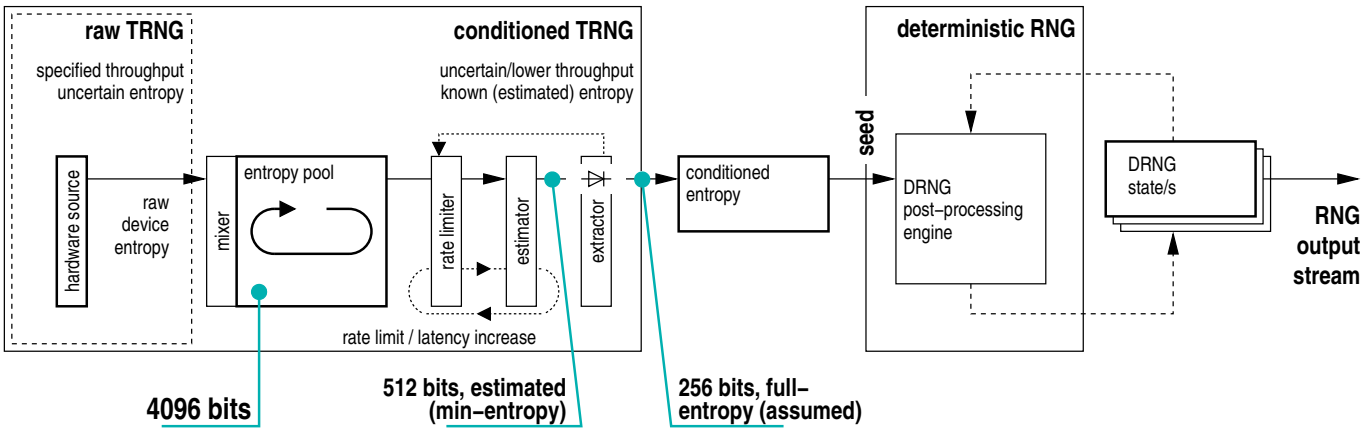

Figure 6: Hybrid random-number generation, data flow

7.1.10 Random-number generation

The TOE shall generate random streams with a cryptographically strong random number generator. The hybrid random-number generation process is based on hardware-provided "true-random" (TRNG) seeds, *conditioned* and then post-processed with a stateful pseudo-random generator (DRNG). Random numbers for direct or indirect use of functional or administrative calls are provided from the DRNG. (SFR FCS_RNG.1)

The TOE obtains TRNG seed from HSM-internal entropy source, "conditioned" by TOE software through a cryptographic hash function — SHA-256 — after min-entropy estimation and rate limiting.

The combination of a conservative entropy-estimator and cryptographic hash function generates "compressed", conditionalized seed blocks in 256-bit increments, which are maintained in an entropy pool aggregating past history. Since the entropy-estimator is conservative, and pool compression to 256 bits is only allowed with 512+ bits of apparent min-entropy, conditionalized TRNG output is assumed to provide 256 full bits of entropy per call (Fig. 6) [BK12, 6.4.2]. Rate limitation prevents entropy compression until at least 512 bytes of the input pool changed since the last compression, preventing iterative guessing of smaller units of entropy [FS03, 10.2] [KSF99, 3.2] [Str16, 3.1].

The pseudo-random generator is based on a non-invertible, cryptographic hash function (SHA-256), instantiating the DRBG structure from [ISO11, C.2.1.1], which conforms to DRG.3 requirements [KS11, 4.9.1] as described in Example 39 of [KS11, 5.6.2]. The DRNG is seeded by maximum-length entropy — 256-bit blocks — from internal, conditioned seed, and maintains state with up to 256 bits of entropy.

Instantiated as a single hybrid RNG [KS11, 4.2], different callers within the TOE obtain their own slices of the generated DRNG stream. TRNG-based reseeding is automatic, and it is not influenced by external entities in the TOE configuration.

7.1.11 Additional functionality

Secure storage *The TOE manages its persistent structures through integrity-checked storage, storing structures as plaintext* under TOE control. *TOE-internal storage is not encrypted by the TOE itself,* but the instantiation form may provide such additional encryption, such as when deployed within IBM HSMs.

Storage controlled by the TOE includes persistent-data structures, heap-allocated global objects, and request-transient stack data. Memory regions which MAY contain sensitive data are wiped at the earliest possible point, before they are released (SFR FDP_RIP.2). (Note that development uses QA tooling specialized to scan memory-lifecycle events for unintended leaks of sensitive data, particularly WKs and blob-cleartext patterns which would be indicative of plaintext not cleared before deallocation.)

Key virtualization Segmentation of keystores based on trust root, application identity, or their combination. Virtualization associates key material with a controlling "session", deriving a unique encryption key for objects bound to each different session. Host-resident state objects or keys reference their controlling session implicitly, as part of their object header. (SFR FIA_USB.1/IU)

Derived encryption keys are discarded upon use, and are not retained within the TOE when the originating request terminates. (Note that the controlling WK and the session identifier are obviously still present in memory, even if any derived key material has been discarded.)

Protection against physical attacks Physical protection is inherited from the enclosure.

Note that we describe a TOE which excludes hardware threats. Tamper-protection capabilities are only mentioned here for completeness; we assume physical protection on a best-effort basis without formal classification. (We acknowledge physical protection afforded by the hosting HSM, but only as an environmental feature.)

8 Support notes

8.1 Specification of full TOE configuration

Configuration identifiers are 256-bit cryptographic hashes (SHA-256).

On IBM model 4767 HSMs, the configuration of segments is as follows:

1. Enterprise PKCS#11: `dada00eb'e58b1075 72deb5e8 426888f4 68d8ef5a 280fe494 2830cac8 97a0c80e`
2. Segment 3: `1dbd97b8'5cae9f30 87f87d18 55bedd3b 23b9cd38 6a93ecbc 2d2a207a 5e573bf5`
3. Segment 2: `49432fd9'3fad4710 dca1ce91 99953275 85eccd8b b4880783 bf2cf36e 64f2991b`
4. Segment 1: `47ded8ee'bb79cf982 250ddbb 1ce945c46 cab4243b d11e4b0d 742664c9 78c1702`

See also Fig. 1. Segment 0 is non-modifiable, therefore its configuration is not included here (it is the same for all 4767 modules).

The above Segment 3 hash implies the EP11 configuration, as this particular EP11 version is embedded within Segment 3. Both are listed here since infrastructure and application counterparties may check the API version, segment configuration, or both.

The equivalent segment configuration for 4765 modules is the following:

1. Enterprise PKCS#11: `e41c1444'e1237584 5880adc4 fb116650 d98f9f42 d3dc3d85 26d3bfbd 6e828212`
2. Segment 3: `c748d76e'7b92860f c15eb45f a720e901 129308e9 c95cb039 6e00e340 9e7a3fd1`
3. Segment 2: `9e64a050'759a13cf dd0b776b 2603c1c4 19efefc2 cdb048ab b50d5495 ba00cf9b`
4. Segment 1: `177caf13'c6012276 90aa8e20 d3bbba58 79a67eba 6c2ad68b 0a3433e0 802c4ea7`

8.2 Function calls

The following top-level calls of PKCS#11 are implemented within EP11, and are available within the TOE:

```
 1  m_Decrypt
 2  m_DecryptFinal
 3  m_DecryptInit
 4  m_DecryptSingle...........addition (DecryptInit + Decrypt)
 5  m_DecryptUpdate

 6  m_DeriveKey
 7  m_Digest
 8  m_DigestFinal
 9  m_DigestInit
10  m_DigestKey

11  m_DigestSingle............addition (DigestInit + Digest)
12  m_DigestUpdate
13  m_Encrypt
14  m_EncryptFinal
15  m_EncryptInit

16  m_EncryptSingle...........addition (EncryptInit + Encrypt)
17  m_EncryptUpdate
18  m_GenerateKey
19  m_GenerateKeyPair
20  m_GenerateRandom

21  m_GetAttributeValue
22  m_GetMechanismInfo
23  m_GetMechanismList
24  m_Login...................modified
25  m_Logout..................modified

26  m_SeedRandom
27  m_SetAttributeValue
28  m_Sign
29  m_SignFinal
30  m_SignInit
```

```
31  m_SignSingle.............addition (SignInit + Sign)
32  m_SignUpdate
33  m_UnwrapKey..............modified
34  m_Verify
35  m_VerifyFinal

36  m_VerifyInit
37  m_VerifySingle...........addition (VerifyInit + Verify)
38  m_VerifyUpdate
39  m_WrapKey................modified
40  m_admin..................addition

41  m_get_xcp_info...........addition
```

Single-pass `nnnSingle()` calls are functionally equivalent to `nnnInit()` immediately followed by a terminating `nnn()` call, without returning intermediate state, for each applicable category of `nnn`. They are supported to reduce the number of roundtrips for this frequently encountered calling pattern; the combined calls provide no additional functionality (but bypass intermediate-state management and one host-module call).

Non-administrative queries specific to EP11 are supplied through a single, non-PKCS#11 query, `get_xcp_info`, with multiple sub-queries. Certain sub-query responses are usable by host code to provide PKCS#11-related information for host libraries, which may be used for simpler transformation between PKCS#11 and EP11 calls — as an example, upper bounds on blob sizes are reported here. Most other sub-queries — such as audit records and history — have no PKCS#11 equivalent. *On-demand algorithm tests may also be invoked as a sub-query.*

While the TOE implementation of session management — `Login` and `Logout` — differs from PKCS#11 due to the different interpretation of sessions and user identities, the interface itself is identical. (The TOE simply maps host-provided authentication data into integrity-checked session identifiers, which may be then exported back to untrusted hosts. These identifiers operate as regular PKCS#11 authentication context otherwise.)

Administrative calls are dispatched from the single `m_admin` service, which — dealing with quantities not present in PKCS#11 itself — has no PKCS#11 equivalent. The service provides both queries and commands.

The addition of attribute-bound transportation modifies the interface of regular PKCS#11 `WrapKey()` and `UnwrapKey()`. Host code may trivially remap regular PKCS#11 calls to our expanded interface; callers of PKCS#11 `Un/WrapKey()` need not be aware of the extension interface (most importantly, the additional authentication key: unlike PKCS#11, attribute-bound transport is encrypted+authenticated).

State-related PKCS#11 calls missing from EP11 have no equivalent in a stateless PKCS#11 implementation, and therefore do not exist within the TOE itself. Host libraries are expected to simulate object-list traversal, session-state queries, and other state-related PKCS#11 services.

8.3 Control points and attributes

Control points restrict specific capabilities within a domain. They are represented in an array of individual CP bits ("CPBs"), with a set bit enabling the capability. Future additions will preserve this positive-active logic, therefore zero-extending a set of CPs from a previous release to a more recent version will remain a safe operation. CPs are administratively managed (SFR FMT_MTD.1/CP).

The following CPBs influence core infrastructure:

- Allow addition/activation of CPBs (`ADD_CPBS`). A separate CPB allows removal of CPBs (`RESTRICT_CPBS`).
  To make the setup read-only, deactivate both of these CPBs. Removing only addition, but not deletion, turns the setup into a form that may be further restricted, but missing capabilities may no longer be activated.
- Allow the backend to save blobs as semi-retained keys (`RETAINKEYS`), which are no longer exportable.
  Note that this setting does not influence key caching, which may not be externally controlled, and is host-opaque.
- Allow modification of object attributes (`MODIFY_OBJECTS`). Removing this attribute makes all objects read-only.
  Note that currently, only Boolean attributes may be modified; non-Boolean attributes including key type, size etc. are never modifiable (SFR FMT_MSA.1/Key_Man_1 vs. SFR FMT_MSA.1/Key_Man_2).
- Allow mixing of external seed to the backend, if it is supported (`RNG_SEED`).
- Allow generating asymmetric keys without selftests upon key generation (`SKIP_KEYTESTS`).

The following control points influence groups of functionality:

- Allow signing with asymmetric (private) keys, symmetric keys (`SIGN_ASYMM`, `SIGN_SYMM`) or verification with symmetric keys (`SIGVERIFY_SYMM`).
  Note that one can not restrict signature verification with public keys, which are available to the host.

- Allow encryption of data with with symmetric keys (ENCRYPT_SYMM), decryption with symmetric or asymmetric ones (DECRYPT_SYMM, DECRYPT_ASYMM).

  Encryption with public keys can not be prevented, therefore there is no such CPB. (Note that *wrapping* keys is separately controlled.)
- Allow generation of symmetric or asymmetric keys (KEYGEN_SYMM, KEYGEN_ASYMM).
- Allow wrapping keys with symmetric or asymmetric keys (WRAP_SYMM, WRAP_ASYMM).
- Allow unwrapping keys with symmetric or asymmetric keys (UNWRAP_SYMM, UNWRAP_ASYMM).
- Allow cryptographic strength windows: below 80 bits, 80 to below-112, 112 to below-127, 128 to below-192, 192 to below-256, or 256-bit (KEYSZ_BELOW80BIT, KEYSZ_80BIT, KEYSZ_112BIT, KEYSZ_128BIT KEYSZ_192BIT, KEYSZ_256BIT).
- Allow algorithms not allowed by NIST or BSI rules of a specific date (ALG_NFIPS2009, ALG_NBSI2009, ALG_NFIPS2011, ALG_NBSI2011)

  These CPBs control entire sets of algorithms. They may be further restricted.

The following CPBs are specific to algorithms or other PKCS#11-level functionality:

- Allow keywrapping without attribute-binding (NON_ATTRBOUND). This CPB must be set for standard PKCS#11 key transport, which uses key forms without attributes.
- Allow raw — unpadded — RSA operations (ALG_RAW_RSA)
- Allow HMAC keys below the minimum FIPS-198 keysize (half of state size) (KEYSZ_HMAC_ANY)
- Allow RSA public keys below $2^{16}+1$ (KEYSZ_RSA65536). This restriction corresponds to the lower limit introduced by FIPS 186–4 (at the end of 2010).
- Allow functional use of RSA, DSA, Diffie-Hellman or elliptic curves (ALG_RSA, ALG_DSA, ALG_DH, ALG_EC)
- Allow EC operations over NIST or Brainpool (E.U.) curves (ALG_EC_NISTCRV, ALG_EC_BPOOLCRV).
- Allow non-administrators to set objects' CKA_TRUSTED attribute, which in turn allows export of keys restricted to transport with trusted keys (USER_SET_TRUSTED). *Note that non-administrator-controlled TRUSTED attributes are inherently unsafe, and need proper privilege separation on the host.*
- Allow creation/use of keys which can un/wrap and en/decrypt simultaneously (WRAP_CRYPT_KEYS). Similar restrictions are possible to prevent sign/verify and and en/decrypt SIGN_CRYPT_KEYS) or un/wrap and sign/verify (WRAP_SIGN_KEYS). The CPs apply to all combinations of symmetric and asymmetric keys.

  These restrictions, when enforced, prevent compromise of key material through misuse of blob attributes, such as mixing encrypted *keys* and *data* [BCFS10, Clu03]. Adding such restrictions prevents the entire category of such attacks.

Control point restrictions are cumulative: all applicable CPs must be enabled for an affected operation to succeed. (SFR FDP_ACF.1/OSA) Error reports indicating a policy-originated rejection are not — currently — specific about which CP has caused the request to be rejected.

8.3.1 Control points list

```
XCP_CPB_ADD_CPBS             0 -- allow addition (activation) of CP bits
XCP_CPB_DELETE_CPBS          1 -- allow removal (deactivation) of CP bits
                               -- remove both ADD_CPBs and DELETE_CPBs
                               -- to make unit read-only
XCP_CPB_SIGN_ASYMM           2 -- sign with private keys
XCP_CPB_SIGN_SYMM........... 3 -- sign with HMAC or CMAC
XCP_CPB_SIGVERIFY_SYMM       4 -- verify with HMAC or CMAC
XCP_CPB_ENCRYPT_SYMM         5 -- encrypt with symmetric keys
                               -- No asymmetric counterpart: one
                               -- may not restrict use of public keys
XCP_CPB_DECRYPT_ASYMM        6 -- decrypt with private keys
XCP_CPB_DECRYPT_SYMM........ 7 -- decrypt with symmetric keys
XCP_CPB_WRAP_ASYMM           8 -- key export with public keys
XCP_CPB_WRAP_SYMM            9 -- key export with symmetric keys
XCP_CPB_UNWRAP_ASYMM        10 -- key import with private keys
XCP_CPB_UNWRAP_SYMM.........11 -- key import with symmetric keys
XCP_CPB_KEYGEN_ASYMM        12 -- generate asymmetric keypairs
XCP_CPB_KEYGEN_SYMM         13 -- generate or derive symmetric keys
                               -- including DSA parameters
XCP_CPB_RETAINKEYS          14 -- allow backend to save semi/retained keys
XCP_CPB_SKIP_KEYTESTS.......15 -- disable selftests on new asymmetric keys
```

```
XCP_CPB_NON_ATTRBOUND       16 -- allow keywrap without attribute-binding
XCP_CPB_MODIFY_OBJECTS      17 -- allow changes to objects (Booleans only)
XCP_CPB_RNG_SEED            18 -- allow mixing external seed to RNG
XCP_CPB_ALG_RAW_RSA.........19 -- allow RSA private-key use without padding
                               -- (highly discouraged)
XCP_CPB_ALG_NFIPS2009       20 -- allow non-FIPS-approved algs (as of 2009)
                               -- including non-FIPS keysizes
XCP_CPB_ALG_NBSI2009        21 -- allow non-BSI algorithms (as of 2009)
                               -- including non-FIPS keysizes
XCP_CPB_KEYSZ_HMAC_ANY      22 -- don't enforce minimum keysize on HMAC
XCP_CPB_KEYSZ_BELOW80BIT....23 -- allow algorithms below 80-bit strength
                               -- public-key operations are still allowed
XCP_CPB_KEYSZ_80BIT         24 -- allow 80  to 111-bit algorithms
XCP_CPB_KEYSZ_112BIT        25 -- allow 112 to 127-bit algorithms
XCP_CPB_KEYSZ_128BIT        26 -- allow 128 to 191-bit algorithms
XCP_CPB_KEYSZ_192BIT........27 -- allow 192 to 255-bit algorithms
XCP_CPB_KEYSZ_256BIT        28 -- allow 256-bit        algorithms
XCP_CPB_KEYSZ_RSA65536      29 -- allow RSA public exponents below 0x10001
XCP_CPB_ALG_RSA             30 -- RSA private-key or key-encrypt use
XCP_CPB_ALG_DSA.............31 -- DSA private-key use
XCP_CPB_ALG_EC              32 -- EC private-key use, see also
                               -- curve restrictions
XCP_CPB_ALG_EC_BPOOLCRV     33 -- Brainpool (E.U.) EC curves
XCP_CPB_ALG_EC_NISTCRV      34 -- NIST/SEC EC curves
XCP_CPB_ALG_NFIPS2011.......35 -- allow non-FIPS-approved algs (as of 2011)
                               -- including non-FIPS keysizes
XCP_CPB_ALG_NBSI2011        36 -- allow non-BSI algorithms (as of 2011)
                               -- including non-BSI keysizes
XCP_CPB_USER_SET_TRUSTED    37 -- allow non-admins to set TRUSTED on a blob/SPKI
XCP_CPB_ALG_SKIP_CROSSCHK   38 -- do not double-check sign/decrypt ops
XCP_CPB_WRAP_CRYPT_KEYS.....39 -- allow keys which can en/decrypt data
                               -- and also un/wrap other keys
XCP_CPB_SIGN_CRYPT_KEYS     40 -- allow keys which can en/decrypt data
                               -- and also sign/verify
XCP_CPB_WRAP_SIGN_KEYS      41 -- allow keys which can un/wrap data
                               -- and also sign/verify
XCP_CPB_USER_SET_ATTRBOUND  42 -- allow non-administrators to
                               -- mark public key objects ATTRBOUND
XCP_CPB_ALLOW_PASSPHRASE....43 -- allow host to pass passphrases, such as
                               -- PKCS12 data, in the clear
XCP_CPB_WRAP_STRONGER_KEY   44 -- allow wrapping of stronger keys
                               -- by weaker keys
XCP_CPB_WRAP_WITH_RAW_SPKI  45 -- allow wrapping with SPKIs without
                               -- MAC and attributes
XCP_CPB_ALG_DH              46 -- Diffie-Hellman use (private keys)
XCP_CPB_DERIVE              47 -- allow key derivation (symmetric+EC/DH)
```

8.3.2 Key/object attributes

The following set of usage restrictions is available within key objects, including encrypted objects and authenticated public keys. Restrictions are generally enforced as their PKCS#11 counterparts are; some additional restrictions are specific to Enterprise PKCS#11 firmware.

```
BLOB_EXTRACTABLE       -- may be encrypted by other keys.
                       -- May not be reset to EXTRACTABLE, once
                       -- made NON-EXTRACTABLE
BLOB_NEVER_EXTRACTABLE -- set if key was created non-extractable.
                       -- Set only initially, may not be modified
BLOB_MODIFIABLE        -- attributes may be changed
BLOB_NEVER_MODIFIABLE  -- object was created read-only.
                       -- Set only initially, may not be modified
BLOB_RESTRICTABLE      -- capabilities may be removed, but may not be
                       -- made more permissive.
BLOB_LOCAL             -- key was created inside this CSP,
```

```
                      -- was not imported.  Set upon object
                      -- creation; may not be modified.
BLOB_ATTRBOUND        -- may be transported only in attribute-bound
                      -- formats, but not standard PKCS11 ones.
                      -- May not be modified.
BLOB_USE_AS_DATA      -- raw key bytes may be input
                      -- to other processing as data,
                      -- such as hashed, or deriving
                      -- keys from them.
BLOB_SIGN             -- may generate signatures
BLOB_SIGN_RECOVER     -- may generate (asymmetric)
                      -- signatures with message recovery
BLOB_DECRYPT
BLOB_ENCRYPT
BLOB_DERIVE
BLOB_UNWRAP           -- may decrypt (transport) other keys
BLOB_WRAP             -- may encrypt (transport) other keys
BLOB_VERIFY
BLOB_VERIFY_RECOVER   -- may verify signatures and recover
                      -- signed messages (asymmetric only)
BLOB_TRUSTED          -- allowed to operate on TRUSTED-only keys
BLOB_WRAP_W_TRUSTED   -- note: _TRUSTED enforcement does not
                      -- provide security guarantees.  We only
                      -- track it inside the HSM to assist hosts.
BLOB_RETAINED         -- backend, not (no longer) on host
BLOB_ALWAYS_RETAINED  -- key has been generated inside, never
                      -- left the hosting module
```

Note that, as noted above, the management of TRUSTED attributes provides no additional security beyond host-based enforcement. The attribute is tracked within object attributes only to simplify host libraries.

8.3.3 Audit event listing

In addition to generic audit categories, the following specific event identifiers are reserved:

```
XCP_LOGSPEV_TRANSACT_ZEROIZE  0xffff0001 -- pending transaction forced module
                                         -- to zeroize (such as: import failed
                                         -- in inconsistent intermediate state)
XCP_LOGSPEV_KAT_FAILED        0xffff0002 -- algorithm known-answer tests failed
XCP_LOGSPEV_KAT_COMPLETED     0xffff0003 -- algorithm known-answer tests passed
XCP_LOGSPEV_EARLY_Q_START.....0xffff0004 -- start of early-audit events:
                                         -- subsequent events have proper order,
                                         -- approximate time
XCP_LOGSPEV_EARLY_Q_END       0xffff0005 -- end of early-audit events:
                                         -- subsequent events include exact time
XCP_LOGSPEV_AUDIT_NEWCHAIN    0xffff0006 -- audit chain was corrupted; removed,
                                         -- generating new instance,
                                         -- starting new chain
XCP_LOGSPEV_TIMECHG_BEFORE    0xffff0007 -- time change: original time
XCP_LOGSPEV_TIMECHG_AFTER.....0xffff0008 -- time change: updated time
XCP_LOGSPEV_MODSTIMPORT_START 0xffff0009 -- accepted full-state import
                                         -- data structure, starting update
XCP_LOGSPEV_MODSTIMPORT_FAIL  0xffff000a -- rejected import structure
                                         -- issued after initial verify
                                         -- indicates some inconsistency
                                         -- of import data structures
XCP_LOGSPEV_MODSTIMPORT_END   0xffff000b -- completed full-state import
```

Note that some of the initial events, since they are issued during startup, where not all of infrastructure is available. As documented in design rationale, these events are logged in chronological order only, but lack certain details of the audit chain state. Event-record regains full context, once all required infrastructure has been initialized.

9 Glossary

Administrator commands are administrative requests requiring authentication—the latter provided through digital signatures.

Blob informal term for host-resident, authenticated-encrypted, opaque, binary objects stored in host-based keystores. These "sensitive" objects are created by EP11 modules; their internals are not host-readable.

CP Control Point, administratively controlled sets of restrictions which enable/disable specific [groups] of functionality

CCPs are *card configuration parameters*, security-critical configuration state of a module, not sensitive. Such critical information includes segment code and ownership (i.e., officer public keys).

CSP Critical security parameter. (To reduce chance of confusion, "cryptographic service provider" is always expanded, never abbreviated in the text.)

Domain name of partitions within EP11 modules. Each domain maintains its own administrative settings and key material.

Driver-based access control on hosts, outside the scope of this security target, is generally based on domains.

EP11 Abbreviation of IBM Enterprise PKCS#11, IBM-specific instantiation of an industry-standard PKCS#11 crypto service provider API

Firmware identifier is an unambiguous, assumed-unique status identifier. We use cryptographic hashes of contents to derive identifiers of at least 256 bits, and assume collisions to be infeasible.

Components are identified by their own segment hashes. Compounds derive identifiers unambiguously. Since we assume unicity of hashes, verifying a top-level compound identifier is sufficient to implicitly identify its constituent components.

FWID Abbreviation of *Firmware identifier*

KAT Known Answer Test

MCPU Module CPU, the processor executing operating system and application code (cf. SSP)

OA *Outbound Authentication*, infrastructure capable of signing by card-resident, non-exportable private keys.

External parties, including other modules, can verify that signed content has been generated by untampered module firmware (Segment 1). An extension allows OA to manage private keys for OS or applications (Segment 2 or 3).

PCIe PCI Express, the external interface of our module.

POST *Power-On Self-Test*, infrastructure tests resident in ROM and flash.

RAS Abbreviation of *Reliability, Availability, Serviceability*

Session The TOE can bind objects to sessions. Users in possession of the valid session PIN can access those objects. The session PIN is therefore considered an authentication means for "session users".

SKI `SubjectKeyIdentifier`, an "assumed-unique" identifier of a public key, generally, a hash of a key-unique parameter. We follow a standard solution, and calculate a hash of the BIT STRING `subjectPublicKey` of the SPKI [RHPFS02, 4.2.1.2]. We currently use SHA-256 as a hash function.

SPKI `SubjectPublicKeyInfo`, a collection of self-describing, industry standard binary formats for public keys. SPKI structures contain type information, unambiguously identifying key types and parameters. RSA SPKIs are described in [SKH05, 1.2], EC ones in [TBY+09, 2.1].

SSP *Security Service Processor*, a dedicated processor executing Miniboot and most of POST (i.e., infrastructure code).

WK Wrapping Key, our terminology for domain-specific keys encrypting an externally stored keystore

References

[Ame01] American Bankers Association. *ANSI x9.63-2001, Elliptic Curve Key Agreement and Key Transport Protocols*, 2001.

[Ame05] American Bankers Association. *ANSI x9.62-2005, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[BB12] William C. Barker and Elaine Barker. *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block cipher (NIST Special Publication 800-67, revision 1)*. National Institute of Standards and Technology (NIST), January 2012.

[BBS06] Daniel Brand, Marcio Buss, and Vugranam C. Sreedhar. Evidence-based analysis and inferring preconditions for bug detection (IBM Research report RC24103). Technical report, IBM Research, October 2006.

[BCFS10] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269, Chicago, Illinois, USA, October 2010. ACM Press.

[BJR+14] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. Using Frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *Proceedings of the 35th IEEE Symposium on Security & Privacy (SOSP'14)*, San Jose, CA, May 2014.

[BK12] Elaine Barker and John Kelsey. *Recommendation for the Entropy Sources Used for Random Bit Generation (NIST Special Publication 800-90B, Draft)*. National Institute of Standards and Technology (NIST), August 2012.

[BK15] Elaine Barker and John Kelsey. *Recommendation for Random Number Generation using Deterministic Random Bit Generators (NIST Special Publication 800-90A, revision 1)*. National Institute of Standards and Technology (NIST), June 2015.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, September 2008.

[Bra00] Daniel Brand. A software falsifier (IBM Research report RC21788). Technical report, IBM Research, October 2000.

[BSI08] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Common Criteria Protection Profile, Cryptographic Modules, Security Level "Enhanced" (BSI-CC-PP-0045-2009, v1.01)*, July 2008.

[Clu03] Jolyon Clulow. On the security of PKCS#11. In *In Proceedings of the 5th International Worshop on Cryptographic Hardware and Embedded Systems (CHES'03), Volume 2779 of LNCS*, pages 411–425. Springer-Verlag, 2003.

[dcd14] Bitcoin developer community discussion. (Bitcoin) checkpoint lockin. online at `en.bitcoin.it/wiki/Checkpoint_Lockin`, June 2014. [accessed 2017-07-09].

[FIP01] National Institute of Standards and Technology (NIST). *Security Requirements for Cryptographic Modules (FIPS 140-2)*, 2001.

[FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

[GIJ+12] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS'12)*, Raleigh, NC, October 2012.

[HD09] Russell Housley and Morris Dworkin. RFC 5649: Advanced encryption standard (AES) key wrap with padding algorithm, August 2009.

[HS02] Russell Housley and Jim Schaad. RFC 3394: Advanced encryption standard (AES) key wrapping algorithm, September 2002.

[ICS12] IBM. *z/OS Cryptographic Services Integrated Cryptographic Service Facility—Writing PKCS#11 Applications (v1R13, SA23-2231-05)*, September 2012.

[ICS16] IBM. *z/OS Cryptographic Services Integrated Cryptographic Service Facility—Administrator's Guide (v2r2, SC14-7506-05)*, April 2016.

[ISO11] ISO/IEC 18031:2011, Information technology, security techniques, random bit generation, 2011.

[JK03] Jakob Jonsson and Burt Kaliski. RFC 3447: Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1, February 2003.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: how secure is SSL?). In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, pages 310–331. Springer-Verlag, 2001.

[KS11] Wolfgang Killmann and Werner Schindler. A proposal for: Functionality classes for random number generators. Technical report, Bundesamt für Sicherheit in der Informationstechnik (BSI), September 2011.

[KSF99] John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator. In *In Sixth Annual Workshop on Selected Areas in Cryptography*, pages 13–33. Springer, 1999.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. RFC 6962: Certificate transparency, June 2013.

[LM10] Manfred Lochter and Johannes Merkle. RFC 5639: Elliptic curve cryptography (ECC) Brainpool standard curves and curve generation, March 2010.

[MA14] Stephen J. Murdoch and Ross Anderson. Security protocols and evidence: Where many payment systems fail. In *Proceedings of Financial Cryptography and Data Security - 18th International Conference, FC'2014, Barbados*, volume 7859 of *Lecture Notes in Computer Science*. Springer, March 2014.

[Mur08] Steven J. Murdoch. Hardened stateless session cookies. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols Workshop*, volume 6615 of *Lecture Notes in Computer Science*, pages 93–101. Springer, 2008.

[Nat99] National Institute of Standards and Technology (NIST). *Data Encryption Standard (FIPS 46–3)*, 1999.

[Nat01a] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard (FIPS 197)*, 2001.

[Nat01b] National Institute of Standards and Technology (NIST). *Recommendation for Block Cipher Modes of Operation (800-38A)*, 2001.

[Nat12] National Institute of Standards and Technology (NIST). *Secure Hash Standard (FIPS 180–4)*, March 2012.

[Nat13a] National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS) (FIPS 186–4)*, July 2013.

[Nat13b] National Institute of Standards and Technology (NIST). *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (SP800-56A rev.2)*, May 2013.

[PKC04] PKCS 11-cryptographic token interface standard (v2.20), June 2004.

[PKC09] PKCS 11 mechanisms v2.30: Cryptoki—draft 7, April 2009.

[PKC14] OASIS. *PKCS 11-Cryptographic Token Interface Standard v2.40, Committee Specification Draft 03*, July 2014.

[RHPFS02] R. R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, April 2002.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, May 1999.

[SKH05] J. Schaad, B. Kaliski, and R. Housley. RFC 4055: Additional algorithms and identifiers for RSA cryptography for use in the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, June 2005.

[Sma13] Nigel P. Smart. Algorithms, key sizes and parameters report – 2013 recommendations. Technical report, EU/ENISA, October 2013.

[SMZ14] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitlodine: extracting intelligence from the Bitcoin network. In *Financial Cryptography and Data Security*, volume (to appear) of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, March 2014.

[Str16] Falko Strenzke. An analysis of OpenSSL's random number generator. Cryptology ePrint Archive, Report 2016/367, `eprint.iacr.org/2016/367`, April 2016. [accessed 2016-05-23].

[TBY+09] S. Turner, D. Brown, K. Yiu, R. Housley, and T. Polk. RFC 5480: Elliptic curve cryptography subject public key information, March 2009.

[TKE16] IBM. *z/OS Cryptographic Services ICSF Trusted Key Entry Workstation User's Guide (version 2r2, SC14-7511-05)*, February 2016.

[VDO14] Tamás Visegrády, Silvio Dragone, and Michael Osborne. Stateless cryptography for virtual environments. *IBM Journal of Research and Development*, 58(1), January 2014.

[Wou12] Karel Wouters. *Hash-chain based protocols for time-stamping and secure logging: formats, analysis and design*. PhD thesis, Katholieke Universiteit Leuven, June 2012.