collaborative Protection Profile for Dedicated Security Component

Version: 1.0
2020-09-10
National Information Assurance Partnership

Revision History

Version	Date	Comment
1.0	2020-09-10	First published release version.
1.0x	2021-04-06	Start of first XML version.

Contents

1 PP introduction
1.1 PP Reference Identification
1.2 Overview
1.3 Terms
1.3.1 Common Criteria Terms
1.3.2 Technical Terms
1.4 Compliant Targets of Evaluation
1.4.1 TOE Boundary
1.4.2 TOE Platform
1.5 Use Cases
2 Conformance Claims
3 Security Problem Description
3.1 Threats
3.2 Assumptions
4 Security Objectives
4.1 Security Objectives for the TOE
4.2 Security Objectives for the Operational Environment
4.3 Security Objectives Rationale
5 Security Requirements
5.1 Security Functional Requirements
5.1.1 Cryptographic Support (FCS)
5.1.2 User Data Protection
5.1.3 Identification and Authentication
5.1.4 TOE Security Functional Requirements Rationale
5.2 Security Assurance Requirements
5.2.1 Class ASE: Security Target
5.2.2 Class ADV: Development
5.2.3 Class AGD: Guidance Documentation
5.2.4 Class ALC: Life-cycle Support
5.2.5 Class ATE: Tests
5.2.6 Class AVA: Vulnerability Assessment
Appendix A - Optional Requirements
A.1 Strictly Optional Requirements
A.2 Objective Requirements
A.3 Implementation-based Requirements
Appendix B - Selection-based Requirements
Appendix C - Inherently Satisfied Requirements
Appendix D - Acronyms
Appendix E - Selection Rules
Appendix F - Use Case Templates
F.1 Elephant-own device
Appendix G - Acronyms
Appendix H - Bibliography

1 PP introduction

1.1 PP Reference Identification

PP Reference: collaborative Protection Profile for Dedicated Security Component

PP Version: 1.0

PP Date: September 10, 2020

1.2 Overview

The scope of this Protection Profile (PP) is to describe the security functionality of QQQQ products in terms of [CC] and to define functional and assurance requirements for such products.

1.3 Terms

The following sections list Common Criteria and technology terms used in this document.

1.3.1 Common Criteria Terms

Assurance	Grounds for confidence that a TOE meets the SFRs [CC].
Base Protection Profile (Base-	Protection Profile used as a basis to build a PP-Configuration.

PP)	
Common Criteria (CC)	Common Criteria for Information Technology Security Evaluation (International Standard ISO/IEC 15408).
Common Criteria Testing Laboratory	Within the context of the Common Criteria Evaluation and Validation Scheme (CCEVS), an IT security evaluation facility, accredited by the National Voluntary Laboratory Accreditation Program (NVLAP) and approved by the NIAP Validation Body to conduct Common Criteria-based evaluations.
Common Evaluation Methodology (CEM)	Common Evaluation Methodology for Information Technology Security Evaluation.
Distributed TOE	A TOE composed of multiple components operating as a logical whole.
Operational Environment (OE)	Hardware and software that are outside the TOE boundary that support the TOE functionality and security policy.
Protection Profile (PP)	An implementation-independent set of security requirements for a category of products.
Protection Profile Configuration (PP-Configuration)	A comprehensive set of security requirements for a product type that consists of at least one Base-PP and at least one PP-Module.
Protection Profile Module (PP-Module)	An implementation-independent statement of security needs for a TOE type complementary to one or more Base Protection Profiles.
Security Assurance Requirement (SAR)	A requirement to assure the security of the TOE.
Security Functional Requirement (SFR)	A requirement for security enforcement by the TOE.
Security Target (ST)	A set of implementation-dependent security requirements for a specific product.
TOE Security Functionality (TSF)	The security functionality of the product under evaluation.
TOE Summary Specification (TSS)	A description of how a TOE satisfies the SFRs in an ST.
Target of Evaluation (TOE)	The product under evaluation.

1.3.2 Technical Terms

Address Space Layout Randomization (ASLR)	An anti-exploitation feature which loads memory mappings into unpredictable locations. ASLR makes it more difficult for an attacker to redirect control to code that they have introduced into the address space of a process.
Administrator	An administrator is responsible for management activities, including setting policies that are applied by the enterprise on the operating system. This administrator could be acting remotely through a management server, from which the system receives configuration policies. An administrator can enforce settings on the system which cannot be overridden by non-administrator users.
Application (app)	Software that runs on a platform and performs tasks on behalf of the user or owner of the platform, as well as its supporting documentation.
Application Programming Interface (API)	A specification of routines, data structures, object classes, and variables that allows an application to make use of services provided by another software component, such as a library. APIs are often provided for a set of libraries included with the platform.
Credential	Data that establishes the identity of a user, e.g. a cryptographic key or password.
Critical Security Parameters (CSP)	Information that is either user or system defined and is used to operate a cryptographic module in processing encryption functions including cryptographic keys and authentication data, such as passwords, the disclosure or modification of which can compromise the security of a cryptographic module or the security of the information protected by the module.

DAR Protection	Countermeasures that prevent attackers, even those with physical access, from extracting data from non-volatile storage. Common techniques include data encryption and wiping.
Data Execution Prevention (DEP)	An anti-exploitation feature of modern operating systems executing on modern computer hardware, which enforces a non-execute permission on pages of memory. DEP prevents pages of memory from containing both data and instructions, which makes it more difficult for an attacker to introduce and execute code.
Developer	An entity that writes OS software. For the purposes of this document, vendors and developers are the same.
General Purpose Operating System	A class of OSes designed to support a wide-variety of workloads consisting of many concurrent applications or services. Typical characteristics for OSes in this class include support for third-party applications, support for multiple users, and security separation between users and their respective resources. General Purpose Operating Systems also lack the real-time constraint that defines Real Time Operating Systems (RTOS). RTOSes typically power routers, switches, and embedded devices.
Host-based Firewall	A software-based firewall implementation running on the OS for filtering inbound and outbound network traffic to and from processes running on the OS.
Operating System (OS)	Software that manages physical and logical resources and provides services for applications. The terms *TOE* and *OS* are interchangeable in this document.
Personally Identifiable Information (PII)	Any information about an individual maintained by an agency, including, but not limited to, education, financial transactions, medical history, and criminal or employment history and information which can be used to distinguish or trace an individual's identity, such as their name, social security number, date and place of birth, mother's maiden name, biometric records, etc., including any other personal information which is linked or linkable to an individual.[OMB]
Sensitive Data	Sensitive data may include all user or enterprise data or may be specific application data such as PII, emails, messaging, documents, calendar items, and contacts. Sensitive data must minimally include credentials and keys. Sensitive data shall be identified in the OS's TSS by the ST author.
User	A user is subject to configuration policies applied to the operating system by administrators. On some systems under certain configurations, a normal user can temporarily elevate privileges to that of an administrator. At that time, such a user should be considered an administrator.
Virtual Machine (VM)	Blah Blah Blah

1.4 Compliant Targets of Evaluation

1.4.1 TOE Boundary

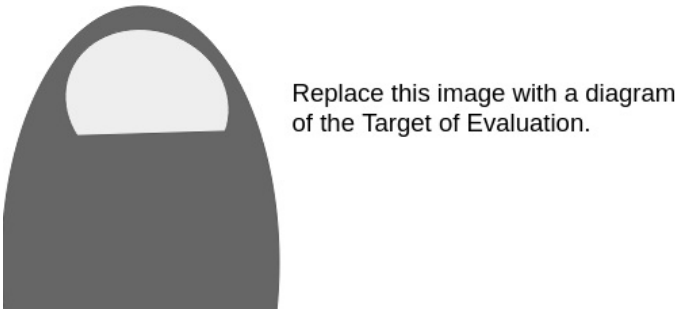

Figure 1: General TOE

1.4.2 TOE Platform

1.5 Use Cases

Requirements in this Protection Profile are designed to address the security problems in at least the following use cases. These use cases are intentionally very broad, as many specific use cases exist for an operating system. These use cases may also overlap with one another. An operating system's functionality may even be effectively extended by privileged applications installed onto it. However, these are out of scope of this PP.

[USE CASE 1] Elephant-own device

This is everything we need to describe in words about this use case.

For a the list of appropriate selections and acceptable assignment values for this configuration, see F.1 Elephant-own device.

2 Conformance Claims

Conformance Statement

An ST must claim exact conformance to this PP, as defined in the CC and CEM addenda for Exact Conformance, Selection-Based SFRs, and Optional SFRs (dated May 2017).

CC Conformance Claims

This PP is conformant to Parts 2 (extended) and 3 (conformant) of Common Criteria Version 3.1, Revision 5.

PP Claim

This PP does not claim conformance to any Protection Profile.

Package Claim

This PP is Functional Package for Transport Layer Security (TLS), version 1.1 Conformant and Functional Package for Secure Shell (SSH), version 1.0 Conformant .

3 Security Problem Description

The security problem is described in terms of the threats that the OS is expected to address, assumptions about the operational environment, and any organizational security policies that the OS is expected to enforce.

3.1 Threats

T.NETWORK_ATTACK

An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may engage in communications with applications and services running on or part of the OS with the intent of compromise. Engagement may consist of altering existing legitimate communications.

T.NETWORK_EAVESDROP

An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between applications and services that are running on or part of the OS.

T.LOCAL_ATTACK

An attacker may compromise applications running on the OS. The compromised application may provide maliciously formatted input to the OS through a variety of channels including unprivileged system calls and messaging via the file system.

T.LIMITED_PHYSICAL_ACCESS

An attacker may attempt to access data on the OS while having a limited amount of time with the physical device.

3.2 Assumptions

A.PLATFORM

The OS relies upon a trustworthy computing platform for its execution. This underlying platform is out of scope of this PP.

A.PROPER_USER

The user of the OS is not willfully negligent or hostile, and uses the software in compliance with the applied enterprise security policy. At the same time, malicious software could act *as* the user, so requirements which confine malicious subjects are still in scope.

A.PROPER_ADMIN

The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4 Security Objectives

4.1 Security Objectives for the TOE

O.ACCOUNTABILITY

Conformant OSes ensure that information exists that allows administrators to discover unintentional issues with the configuration and operation of the operating system and discover its cause. Gathering event information and immediately transmitting it to another system can also enable incident response in the event of system compromise.

O.INTEGRITY

Conformant OSes ensure the integrity of their update packages. OSes are seldom if ever shipped without errors, and the ability to deploy patches and updates with integrity is critical to enterprise network security. Conformant OSes provide execution environment-based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems.

O.MANAGEMENT

To facilitate management by users and the enterprise, conformant OSes provide consistent and supported interfaces for their security-relevant configuration and maintenance. This includes the deployment of applications and application updates through the use of platform-supported deployment mechanisms and formats, as well as providing mechanisms for configuration and application execution control.

O.PROTECTED_STORAGE

To address the issue of loss of confidentiality of credentials in the event of loss of physical control of the storage medium, conformant OSes provide data-at-rest protection for credentials. Conformant OSes also provide access controls which allow users to keep their files private from other users of the same system.

O.PROTECTED_COMMS

To address both passive (eavesdropping) and active (packet modification) network attack threats, conformant OSes provide mechanisms to create trusted channels for CSP and sensitive data. Both CSP and sensitive data should not be exposed outside of the platform.

4.2 Security Objectives for the Operational Environment

The following security objectives for the operational environment assist the OS in correctly providing its security functionality. These track with the assumptions about the environment.

OE.PLATFORM

The OS relies on being installed on trusted hardware.

OE.PROPER_USER
The user of the OS is not willfully negligent or hostile, and uses the software within compliance of the applied enterprise security policy. Standard user accounts are provisioned in accordance with the least privilege model. Users requiring higher levels of access should have a separate account dedicated for that use.

OE.PROPER_ADMIN
The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4.3 Security Objectives Rationale

This section describes how the assumptions, threats, and organization security policies map to the security objectives.

Table 1: Security Objectives Rationale

Threat, Assumption, or OSP	Security Objectives	Rationale
T.NETWORK_ATTACK	O.PROTECTED_COMMS	The threat T.NETWORK_ATTACK is countered by O.PROTECTED_COMMS as this provides for integrity of transmitted data.
	O.INTEGRITY	The threat T.NETWORK_ATTACK is countered by O.INTEGRITY as this provides for integrity of software that is installed onto the system from the network.
	O.MANAGEMENT	The threat T.NETWORK_ATTACK is countered by O.MANAGEMENT as this provides for the ability to configure the OS to defend against network attack.
	O.ACCOUNTABILITY	The threat T.NETWORK_ATTACK is countered by O.ACCOUNTABILITY as this provides a mechanism for the OS to report behavior that may indicate a network attack has occurred.
T.NETWORK_EAVESDROP	O.PROTECTED_COMMS	The threat T.NETWORK_EAVESDROP is countered by O.PROTECTED_COMMS as this provides for confidentiality of transmitted data.
	O.MANAGEMENT	The threat T.NETWORK_EAVESDROP is countered by O.MANAGEMENT as this provides for the ability to configure the OS to protect the confidentiality of its transmitted data.
T.LOCAL_ATTACK	O.INTEGRITY	The objective O.INTEGRITY protects against the use of mechanisms that weaken the TOE with regard to attack by other software on the platform.
	O.ACCOUNTABILITY	The objective O.ACCOUNTABILITY protects against local attacks by providing a mechanism to report behavior that may indicate a local attack is occurring or has occurred.
T.LIMITED_PHYSICAL_ACCESS	O.PROTECTED_STORAGE	The objective O.PROTECTED_STORAGE protects against unauthorized attempts to access physical storage used by the TOE.
A.PLATFORM	OE.PLATFORM	The operational environment objective OE.PLATFORM is realized through A.PLATFORM.
A.PROPER_USER	OE.PROPER_USER	The operational environment objective OE.PROPER_USER is realized through A.PROPER_USER.
A.PROPER_ADMIN	OE.PROPER_ADMIN	The operational environment objective OE.PROPER_ADMIN is realized through A.PROPER_ADMIN.

5 Security Requirements

This chapter describes the security requirements which have to be fulfilled by the product under evaluation. Those requirements comprise functional components from Part 2 and assurance components from Part 3 of [CC]. The following conventions are used for the completion of operations:

- **Refinement** operation (denoted by **bold text** or ~~strikethrough text~~): is used to add details to a requirement (including replacing an assignment with a more restrictive selection) or to remove part of the requirement that is made irrelevant through the completion of another operation, and thus further restricts a requirement.
- **Selection** (denoted by *italicized text*): is used to select one or more options provided by the [CC] in stating a requirement.
- **Assignment** operation (denoted by *italicized text*): is used to assign a specific value to an unspecified parameter, such as the length of a password. Showing the value in square brackets indicates assignment.
- **Iteration** operation: is indicated by appending the SFR name with a slash and unique identifier suggesting the purpose of the operation, e.g. "/EXAMPLE1."

5.1 Security Functional Requirements

5.1.1 Cryptographic Support (FCS)

FCS_CKM.1 Cryptographic Key Generation

FCS_CKM.1.1

The TSF shall generate cryptographic keys by [*parsing in accordance with FDP_ITC_EXT.1 and FDP_ITC_EXT.2*, [**selection**: *asymmetric key generation in accordance with FCS_CKM.1/AK, symmetric key generation in accordance to FCS_CKM.1/SK, no other methods*] ~~in accordance with a specified cryptographic key generation algorithm [assignment: *cryptographic key generation algorithm*] and specified cryptographic key sizes [assignment: *cryptographic key sizes*] that meet the following: [assignment: *list of standards*]~~.

Application Note: Parsing of keys can refer to both the act of importing keys from outside the TOE boundary and to the act of issuing commands or parameters to the TOE that trigger the TSF to perform a key generation function.

If asymmetric key generation in accordance with FCS_CKM.1/AK is selected, the selection-based SFR FCS_CKM.1/AK must be claimed by the TOE.

If symmetric key generation in accordance with FCS_CKM.1/SK is selected, the selection-based SFR FCS_CKM.1/SK must be claimed by the TOE.

TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains a cryptographic key through importation of keys from external sources as specified in FDP_ITC_EXT.1 and FDP_ITC_EXT.2. The evaluator shall also examine the TSS to determine whether it describes any supported asymmetric or symmetric key generation functionality consistent with the claims made in FCS_CKM.1.1.
Guidance
The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST. The evaluator shall confirm that the KMD describes:

- The parsing interface and how the TSF imports keys for internal use
- The asymmetric key generation interfaces and how the TSF internally creates asymmetric keys, if claimed
- The symmetric key generation interfaces and how the TSF internally creates symmetric keys, if claimed

If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the key is used as part of the key chain/hierarchy.
Tests
Testing for this function is performed in conjunction with FDP_ITC_EXT.1 and FDP_ITC_EXT.2. If asymmetric or symmetric key generation functionality is claimed, testing for this function is also performed in conjunction with FCS_CKM.1/AK or FCS_CKM.1/SK.

Evaluation Activities

FCS_CKM.1:
TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains a cryptographic key through importation of keys from external sources as specified in FDP_ITC_EXT.1 and FDP_ITC_EXT.2. The evaluator shall also examine the TSS to determine whether it describes any supported asymmetric or symmetric key generation functionality consistent with the claims made in FCS_CKM.1.1.
Guidance
The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST. The evaluator shall confirm that the KMD describes:

- *The parsing interface and how the TSF imports keys for internal use*
- *The asymmetric key generation interfaces and how the TSF internally creates asymmetric keys, if claimed*
- *The symmetric key generation interfaces and how the TSF internally creates symmetric keys, if claimed*

If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the key is used as part of the key chain/hierarchy.
Tests
Testing for this function is performed in conjunction with FDP_ITC_EXT.1 and FDP_ITC_EXT.2. If asymmetric or symmetric key generation functionality is claimed, testing for this function is also performed in conjunction with FCS_CKM.1/AK or FCS_CKM.1/SK.

FCS_CKM.1/AK Cryptographic Key Generation (Asymmetric Keys)

FCS_CKM.1.1/AK

The TSF shall generate **asymmetric** cryptographic keys using the methods defined by the following rows in Table 2: [**selection**: *AK1*, *AK2*, *AK3*, *AK4*, *AK5*].

Table 2: Supported Methods for Asymmetric Key Generation

Identifier	Key Type	Key Sizes	List of Standards
AK1	RSA	[**selection**: *2048 bit*, *3072-bit*]	FIPS PUB 186-4 (Section B.3)
AK2	ECC-N	[**selection**: *256 (P-256)*, *384 (P-384)*, *521 (P-521)*]	FIPS PUB 186-4 (Section B.4 & D.1.2)
AK3	ECC-B	[**selection**: *256 (brainpoolP256r1)*, *384 (brainpoolP384r1)*, *512 (brainpoolP512r1)*]	RFC5639 (Section 3) (Brainpool Curves)
AK4	DSA	DSA Bit lengths of p and q respectively (L, N) [**selection**: *(1024, 160)*, *(2048, 224)*, *(2048, 256)*, *(3027, 256)*]	FIPS 186-4 Appendix B.1
AK5	Curve25519	256 bits	RFC 7748

Application Note: This requirement is included for the purposes of encryption and decryption operations only. To support ITE protected communications requirement for the transfer of encrypted data, this requirement mandates implementation compliance to FIPS 186-4 only. Implementations according to FIPS 186-2 or FIPS 186-3 will not be accepted.

This requirement must be claimed by the TOE if at least one of FCS_CKM.1 or FCS_CKM.1/KEK chooses a selection related to generation of asymmetric keys.

Evaluation Activities

FCS_CKM.1/AK:

FCS_CKM.1/SK Cryptographic Key Generation (Symmetric Encryption Key)

FCS_CKM.1.1/SK

The TSF shall generate **symmetric** cryptographic keys using the methods defined by the following rows in Table 3: [**selection**: *RSK*, *DSK*, *PBK*].

Table 3: Supported Methods for Symmetric Key Generation

Identifier	Key Type	Cryptographic Key Generation Algorithm	Key Sizes	List of Standards
RSK	[**selection**: *symmetric key*, *submask*, *authorization value*]	Direct Generation from a Random Bit Generator as specified in FCS_RBG_EXT.1	[**selection**: *128*, *192*, *256*, *512*] bits	NIST SP 800-133 (Section 7.1) with ISO 18031 as an approved RBG in addition to those in NIST SP 800-133 (Section 5).
DSK [**selection**: *identifier from Table 16: Key Derivation Functions*]	[**selection**: *Key Type from Table 16: Key Derivation Functions*]	Derived from a Key Derivation Function as specified in FCS_CKM_EXT.5 [**selection**: *Key Derivation Algorithm from Table 16: Key Derivation Function*]	[**selection**: *key sizes from Table 16: Key Derivation Functions*]	[**selection**: *List of Standards from Table 16: Key Derivation Functions*]
PBK	[**selection**: *submask*, *authentication token*, *authorization value*]	Derived from a Password Based Key Derivation Function as specified in FCS_COP.1/PBKDF	[**selection**: *key sizes as specified in FCS_COP.1/PBKDF*]	[**selection**: *standards as specified in FCS_COP.1/PBKDF*]

Application Note: The intent of this requirement is to ensure that attackers cannot recover SKs with less than a full exhaust of the key space. This requirement explains SK generation regardless of how the DSC uses it or when it generates it. The encryption of user data that is not keying material, authentication tokens, or authorization values is outside the scope of this cPP. This cPP assumes that the DSC lacks the required resources to perform bulk encryption/decryption services at a suitable rate for users. The host may use the SK for encrypting user data outside the boundaries of the DSC. On the other hand, the DSC may use the SK on behalf of the user to perform keyed hashes. In this case, all the requirements for generating, controlling access and use, and destroying the key while under the protection of the DSC apply. The selection of key size 512 bits is for the case of XTS-AES using AES-256. In the case of XTSAES for both AES-128 and AES-256, the developer is expected to ensure that the full key is generated using direct generation from the RBG as in NIST SP 800-133 section.

The ST author selects at least one algorithm from the RSK row if the ST supports creating keys directly from the output of the RBG without further conditioning, at least one algorithm from the DSK row should be selected if the ST supports key derivation functions which are usually seeded from RBG and then further conditioned to the appropriate key size, and at least one algorithm from the PBK row should be selected if the ST supports keys derived from passwords.

If DSK is selected, the selection-based SFR FCS_CKM_EXT.5 must be claimed by the TOE.

If PBK is selected, the selection-based SFR FCS_COP.1/PBKDF must be claimed by the TOE.

This requirement must be claimed by the TOE if at least one of FCS_CKM.1 or FCS_CKM.1/KEK chooses a selection related to generation of symmetric keys.

Evaluation Activities

FCS_CKM.1/SK:

FCS_CKM.1/KEK Cryptographic Key Generation (Key Encryption Key)

FCS_CKM.1.1/KEK

The TSF shall generate key encryption keys in accordance with a specified cryptographic key generation algorithm corresponding to [**selection**:

- *Asymmetric KEKs generated in accordance with FCS_CKM.1/AK identifier AK1*,
- *Symmetric KEKs generated in accordance with FCS_CKM.1/SK*,
- *Derived KEKs generated in accordance with FCS_CKM_EXT.5*

] and specified cryptographic key sizes [**assignment**: *cryptographic key sizes*] that meet the following: [**assignment**: *list of standards*].

Application Note: KEKs protect KEKs and Symmetric Keys (SKs). DSCs should use key strengths commensurate with protecting the chosen symmetric encryption key strengths. If Asymmetric KEKs generated in accordance with FCS_CKM.1/AK is selected, the selection-based SFR FCS_CKM.1/AK must be claimed by the TOE.

If Symmetric KEKs generated in accordance with FCS_CKM.1/SK is selected, the selection-based SFR FCS_CKM.1/SK must be claimed by the TOE.

If Derived KEKs generated in accordance with FCS_CKM_EXT.5 is selected, the selection-based SFR FCS_CKM_EXT.5 must be claimed by the TOE.

Evaluation Activities

FCS_CKM.1/KEK:
TSS
The evaluator shall examine the key hierarchy section of the TSS to ensure that the formation of all KEKs is described and that the key sizes match that described by the ST author. The evaluator shall examine the key hierarchy section of the TSS to ensure that each KEK encrypts keys of equal or lesser security strength using one of the selected methods.

[conditional] If the KEK is generated according to an asymmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/AK is invoked. The evaluator uses the description of the key generation functionality in FCS_CKM.1/AK or documentation available for the operational environment to determine that the key strength being requested is greater than or equal to 112 bits.

[conditional] If the KEK is generated according to a symmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/SK is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1, or the key derivation functionality in either FCS_CKM_EXT.5 or FCS_COP.1/PBKDF, depending on the key generation method claimed, to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.

[conditional] If the KEK is formed from derivation, the evaluator shall verify that the TSS describes the method of derivation and that this method is consistent with FCS_CKM_EXT.5.

Guidance

There are no guidance evaluation activities for this component. The evaluator shall iterate through each of the methods selected by the ST and confirm that the KMD describes the applicable selected methods.
Tests
The evaluator shall iterate through each of the methods selected by the ST and perform all applicable tests from the selected methods.

FCS_CKM.2 Cryptographic Key Establishment

FCS_CKM.2.1

The TSF shall establish cryptographic keys in accordance with a specified cryptographic key establishment method: [**selection**:

- *RSA-based key establishment schemes that meet the following: NIST Special Publication 800-56B Revision 2, "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography",*
- *RSA-based key establishment schemes that meet the following: RSAES-PKCS1-v1_5 as specified in Section 7.2 of RFC 8017, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2",*
- *Elliptic curve-based key establishment schemes that meet the following: [**selection**:*
  - *NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",*
  - *RFC 7748, "Elliptic Curves for Security"*

  ],
- *Finite field-based key establishment schemes that meet the following: NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",*
- *Elliptic Curve Integrated Encryption Scheme (ECIES) that meets the following: [**selection**:*
  - *ANSI X9.63 - Public Key Cryptography for the Financial Services Industry Key Agreement and Key Transport Using Elliptic Curve Cryptography,*
  - *IEEE 1363a - Standard Specification for Public-Key Cryptography - Amendment 1: Additional Techniques,*
  - *ISO/IEC 18033-2 - Information Technology - Security Techniques - Encryption Algorithms - Part 2: Asymmetric Ciphers,*
  - *SECG SEC1 - Standards for Efficient Cryptography Group Elliptic Curve Cryptography, section 5.1 Elliptic Curve Integrated Encryption Scheme*

  ]

] ~~that meets the following: [*assignment: list of standards*]~~.

Application Note: This is a refinement of the SFR FCS_CKM.2 to deal with key establishment rather than key distribution.

The ST author selects all key establishment schemes used for the selected cryptographic protocols.

The RSA-based key establishment schemes are described in Section 8 of NIST SP 800-56B Revision 2 [NIST-RSA]; however, Section 8 relies on implementation of other sections in SP 800-56B Revision 2.

The elliptic curves used for the key establishment scheme correlate with the curves specified in FCS_CKM.1/AK.

The selections in this SFR must be consistent with those for FCS_COP.1/KAT.

Evaluation Activities

FCS_CKM.2:
TSS
The evaluator shall examine the TSS to ensure that ST supports at least one key establishment scheme. The evaluator also ensures that for each key establishment scheme selected by the ST in FCS_CKM.2.1 it also supports one or more corresponding methods selected in FCS_COP.1/KAT. If the ST selects RSA in FCS_CKM.2.1, then the TOE must support one or more of "KAS1," or "KAS2," "KTS-OAEP," from FCS_COP.1/KAT. If the ST selects elliptic curve-based, then the TOE must support one or more of "ECDH-NIST" or "ECDH-BPC" from FCS_COP.1/KAT. If the ST selects Diffie-Hellman-based key establishment, then the TOE must support "DH" from FCS_COP.1/KAT.

Guidance
The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme. There are no KMD evaluation activities for this component.
Tests
Testing for this SFR is performed under the corresponding functions in FCS_COP.1/KAT.

FCS_CKM.4 Cryptographic Key Destruction

FCS_CKM.4.1

The TSF shall destroy cryptographic keys and keying material in accordance

with a specified cryptographic key destruction method

- For volatile memory, the destruction shall be executed by a [**selection**:
  - *single overwrite consisting of [**selection**: a pseudo-random pattern using the TSF's RBG, zeroes, ones, a new value of a key, [**assignment**: some value that does not contain any CSP]],*
  - *removal of power to the memory,*
  - *removal of all references to the key directly followed by a request for garbage collection*

  ]
- For non-volatile memory [**selection**:
  - *that employs a wear-leveling algorithm, the destruction shall be executed by a [**selection**:*
    - *single overwrite consisting of [**selection**: zeroes, ones, pseudo-random pattern, a new value of a key of the same size, [**assignment**: some value that does not contain any CSP]],*
    - *block erase*

    ],
  - *that does not employ a wear-leveling algorithm, the destruction shall be executed by a [**selection**:*
    - *[**selection**: single, [**assignment**: ST author-defined multi-pass]] overwrite consisting of [**selection**: zeros, ones, pseudo-random pattern, a new value of a key of the same size, [**assignment**: some value that does not contain any CSP]] followed by a read-verify. If the read-verification of the overwritten data fails, the process shall be repeated again up to [**assignment**: number of times to attempt overwrite] times, whereupon an error is returned.,*
    - *block erase*

    ]

  ]

that meets the following: [*no standard*].

Application Note: A DSC must implement mechanisms to destroy cryptographic keys and key material contained in persistent storage when no longer needed. The term "cryptographic keys" in this SFR includes the authorization data that is the entry point to a key chain and all other cryptographic keys and keying material (whether in plaintext or encrypted form). This SFR does not apply to the public component of asymmetric key pairs, or to keys that are permitted to remain stored such as device identification keys.

In the case of volatile memory, the selection "removal of all references to the key directly followed by a request for garbage collection" is used in a situation where the TSF cannot address the specific physical memory locations holding the data to be erased and therefore relies on addressing logical addresses (which frees the relevant physical addresses holding the old data) and then requesting the platform to ensure that the data in the physical addresses is no longer available for reading (i.e. the "garbage collection" referred to in the SFR text). Guidance documentation for the TOE requires users not to allow the TOE to leave the user's control while a session is active (and hence while the DEK is likely to be in plaintext in volatile memory).

The selection for destruction of data in non-volatile memory includes block erase as an option, and this option applies only to flash memory. A block erase does not require a read verify, since collaborative Protection Profile for Dedicated Security Components the mappings of logical addresses to the erased memory locations are erased as well as the data itself.

Where different destruction methods are used for different data or different destruction situations then the different methods and the data/situations they apply to (e.g. different points in time, or power-loss situations) are described in the TSS (and the ST may use separate iterations of the SFR to aid clarity). The TSS includes a table describing all relevant keys and keying material (including authorization data) used in the implementation of the SFRs, stating the source of the data, all memory types in which the data is stored (covering storage both during and outside of a session, and both plaintext and non-plaintext forms of the data), and the applicable destruction method and time of destruction in each case.

Some selections allow assignment of "some value that does not contain any CSP." This means that the TOE uses some specified data not drawn from an RBG meeting FCS_RBG_EXT requirements, and not being any of the particular values listed as other selection options. The point of the phrase "does not contain any sensitive data" is to ensure that the overwritten data is carefully selected, and not taken from a general pool that might contain current or residual data (e.g. SDOs or intermediate key chain values) that itself requires confidentiality protection.

Evaluation Activities

FCS_CKM.4:
TSS
The evaluator shall examine the TSS to ensure it lists all relevant keys and keying material

(describing the source of the data, all memory types in which the data is stored (covering storage both during and outside of a session, and both plaintext and non-plaintext forms of the data)), all relevant destruction situations (including the point in time at which the destruction occurs; e.g. factory reset or device wipe function, change of authorization data, change of DEK, completion of use of an intermediate key) and the destruction method used in each case. The evaluator shall confirm that the description of the data and storage locations is consistent with the functions carried out by the TOE (e.g. that all keys in the key chain are accounted for). (Where keys are stored encrypted or wrapped under another key then this may need to be explained in order to allow the evaluator to confirm the consistency of the description of keys with the TOE functions).

The evaluator shall check that the TSS identifies any configurations or circumstances that may not conform to the key destruction requirement (see further discussion in the AGD section below). Note that reference may be made to the AGD for description of the detail of such cases where destruction may be prevented or delayed.

Where the ST specifies the use of "a value that does not contain any sensitive data" to overwrite keys, the evaluator shall examine the TSS to ensure that it describes how that pattern is obtained and used, and that this justifies the claim that the pattern does not contain any sensitive data.

Guidance
The evaluator shall check that the guidance documentation for the TOE requires users to ensure that the TOE remains under the user's control while a session is active.

A TOE may be subject to situations that could prevent or delay data destruction in some cases. The evaluator shall check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS (and KMD). The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer, identifying any additional mitigation actions for the user (e.g. there might be some operation the user can invoke, or the user might be advised to retain control of the device for some particular time to maximise the probability that garbage collection will have occurred).

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may implement wear-levelling and garbage collection. This may result in additional copies of the data that are logically inaccessible but persist physically. Where available, the TOE might then describe use of the TRIM command and garbage collection to destroy these persistent copies upon their deletion (this would be explained in TSS and guidance documentation).

Where TRIM is used then the TSS or guidance documentation is also expected to describe how the keys are stored such that they are not inaccessible to TRIM, (e.g. they would need not to be contained in a file less than 982 bytes which would be completely contained in the master file table.

The evaluator shall examine the KMD to verify that it identifies and describes the interfaces that are used to service commands to read/write memory. The evaluator shall examine the interface description for each different media type to ensure that the interface supports the selections made by the ST author.

45 The evaluator shall examine the KMD to ensure that all keys and keying material identified in the TSS and KMD have been accounted for.

46 Note that where selections include 'destruction of reference to the key directly followed by a request for garbage collection' (for volatile memory) then the evaluator shall examine the KMD to ensure that it explains the nature of the destruction of the reference, the request for garbage collection, and of the garbage collection process itself.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform the following tests:

- ***Test 1:*** *Applied to each key or keying material held as plaintext in volatile memory and subject to destruction by overwrite by the TOE (whether or not the plaintext value is subsequently encrypted for storage in volatile or non-volatile memory).*

  The evaluator shall:
  1. *Record the value of the key or keying material.*
  2. *Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.*
  3. *Search the content of the binary file created in Step #2 to locate all instances of the known key value from Step #1.*

     Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #8 and #9.
  4. *Cause the TOE to perform normal cryptographic processing with the key from Step #1.*
  5. *Cause the TOE to destroy the key.*
  6. *Cause the TOE to stop execution but not exit.*
  7. *Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.*
  8. *Search the content of the binary file created in Step #7 for instances of the known key value from Step #1.*
  9. *Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search.)*

Steps #1-8 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

Step #9 ensures that partial key fragments do not remain in memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.

- ***Test 2:*** *Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.*
  1. *Record the value of the key or keying material.*
  2. *Cause the TOE to perform normal cryptographic processing with the key from Step #1.*
  3. *Search the non-volatile memory the key was stored in for instances of the known key value from Step #1.*

     Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #5 and #6.
  4. *Cause the TOE to clear the key.*
  5. *Search the non-volatile memory in which the key was stored for instances of the known key value from Step #1. If a copy is found, then the test fails.*
  6. *Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search).*

  Step #6 ensures that partial key fragments do not remain in non-volatile memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.
- ***Test 3:*** *Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.*
  1. *Record memory of the key or keying material.*
  2. *Cause the TOE to perform normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to clear the key. Record the value to be used for the overwrite of the key.*
  4. *Examine the memory from Step #1 to ensure the appropriate pattern (recorded in Step #3) is used.*

  The test succeeds if correct pattern is found in the memory location. If the pattern is not found, then the test fails.

FCS_CKM_EXT.4 Cryptographic Key and Key Material Destruction Timing

FCS_CKM_EXT.4.1

The TSF shall destroy all keys and keying material when no longer needed.

Application Note: The DSC will have mechanisms to destroy keys, including intermediate keys and key material, by using an approved method, FCS_CKM.4. Examples of keys include intermediate keys, leaf keys, encryption keys, signing keys, verification keys, authentication tokens, and submasks. The DSC will have mechanisms to destroy keys and key material contained in persistent storage when no longer needed. Based on their implementation, vendors will explain when certain keys are no longer needed. An example in which key is no longer necessary includes a wrapped key whose password has changed. However, there are instances when keys are allowed to remain in memory, for example, a device identification key.

Evaluation Activities

FCS_CKM_EXT.4:
TSS
The evaluator shall verify the TSS provides a high-level description of what it means for keys and key material to be no longer needed and when this data should be expected to be destroyed.
Guidance
There are no guidance evaluation activities for this component. The evaluator shall verify that the KMD includes a description of the areas where keys and key material reside and when this data is no longer needed.

The evaluator shall verify that the KMD includes a key lifecycle that includes a description where key materials reside, how the key materials are used, how it is determined that keys and key material are no longer needed, and how the data is destroyed once it is no longer needed. The evaluator shall also verify that all key destruction operations are performed in a manner specified by FCS_CKM.4.

Tests
There are no test evaluation activities for this component

FCS_CKM_EXT.5 Cryptographic Key Derivation

FCS_CKM_EXT.5.1

The TSF shall generate cryptographic keys using the Key Derivation Functions defined by the following rows of Table 4: [**selection**: *KeyDrv1*, *KeyDrv2*, *KeyDrv3*, *KeyDrv4*, *KeyDrv5*, *KeyDrv6*, *KeyDrv7*, *KeyDrv8*].

Table 4: Key Derivation Functions

Identifier	Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Stan
KeyDrv1	[**selection**: *symmetric key*, *initialization vector*, *authentication token*, *authorization value*, *HMAC key*, *KMAC key*]	Direct Generation from a Random Bit Generator as specified in FCS_RBG_EXT.1	KDF in Counter Mode using [**selection**: *CMAC-AES-128*, *CMAC-AES-192*, *CMAC-AES-256*, *HMAC-SHA-1*, *HMAC-SHA-256*, *HMAC-SHA-512*]as the PRF	[**selection**: *128*, *192*, *256*]bits	NIST SP 800 (Section 5.1) in Counter M [**selection**: *CMAC*, *NIST CMAC*, *ISO- ISO-HMAC*, *HMAC*, *ISO- FIPS-SHA*]
KeyDrv2	[**selection**: *symmetric key*, *initialization vector*, *authentication token*, *authorization value*, *HMAC key*, *KMAC key*]	Direct Generation from a Random Bit Generator as specified in FCS_RBG_EXT.1	KDF in Feedback Mode using [**selection**: *CMAC-AES-128*, *CMAC-AES-192*, *CMAC-AES-256*, *HMAC-SHA-1*, *HMAC-SHA-256*, *HMAC-SHA-512*]as the PRF	[**selection**: *128*, *192*, *256*]bits	NIST SP 800 (Section 5.2) in Feedback [**selection**: *CMAC*, *NIST CMAC*, *ISO- ISO-HMAC*, *HMAC*, *ISO- FIPS-SHA*]
KeyDrv3	[**selection**: *symmetric key*, *initialization vector*, *authentication token*, *authorization value*, *HMAC key*, *KMAC key*]	Direct Generation from a Random Bit Generator as specified in FCS_RBG_EXT.1	KDF in Double Pipeline Iteration Mode using [**selection**: *CMAC-AES-128*, *CMAC-AES-192*, *CMAC-AES-256*, *HMAC-SHA-1*, *HMAC-SHA-256*, *HMAC-SHA-512*]as the PRF	[**selection**: *128*, *192*, *256*]bits	NIST SP 800 (Section 5.3) in n Double Pipeline Iter Mode) [**selection**: *CMAC*, *NIST CMAC*, *ISO- ISO-HMAC*, *HMAC*, *ISO- FIPS-SHA*]
KeyDrv4	[**selection**: *symmetric key*, *initialization vector*, *authentication token*, *authorization value*, *HMAC key*, *KMAC key*]	Intermediary keys	[**selection**: *exclusive OR (XOR)*, *SHA256*, *SHA-512*]	[**selection**: *128*, *192*, *256*]bits	[**selection**: *HASH*, *FIPS*
KeyDrv5	[**selection**: *symmetric key*, *initialization vector*, *authentication token*, *authorization value*, *HMAC key*, *KMAC key*]	Concatenated keys	KDF in [**selection**: *Counter Mode*, *Feedback Mode*, *Double Pipeline Iteration Mode*] using [**selection**: *CMAC-AES-128*, *CMAC-AES-192*, *CMAC-AES-256*, *HMAC-SHA-1*, *HMAC-SHA-256*, *HMAC-SHA-512*]as the PRF	[**selection**: *128*, *192*, *256*]bits	NIST SP 800 [selection: (S 5.1) (KDF in Counter Mo (Section 5.2) in Feedback Mode); (Section 5.3) in Double-Pi Iteration Mo [**selection**: *CMAC*, *NIST CMAC*, *ISO- ISO-HMAC*, *HMAC*, *ISO- FIPS-SHA*]
KeyDrv6	[**selection**: *symmetric key*, *initialization vector*, *authentication*	Two keys	[**selection**: *AES-CCM*, *AES-GCM*, *AES-CBC*, *AES-KWP*, *AES-KW*, *CAM-CBC*, *CAM-CCM*, *CAM-GCM*]	[**selection**: *128*, *192*, *256*]bits	[selection: s of Standards FCS_COP.1/ Symmetric K table]

	token, *authorization value*, *HMAC key*, *KMAC key*]		from FCS_COP.1/SKC Symmetric Key table		
KeyDrv7	[**selection**: *symmetric key*, *secret IV*, *seed*]	Shared secret, salt, output length, fixed information	[**selection**: *hash function from FCS_COP.1/Hash*, *keyed hash from FCS_COP.1/HMAC*]	[**selection**: *128*, *192*, *256*]bits	(NIST-KDRV [**selection**: *List of Stand in FCS_COP.1/ and FCS_COP.1/*
KeyDrv8	[**selection**: *symmetric key*, *secret IV*, *seed*]	Shared secret, salt, IV, output length, fixed information	[**selection**: *keyed hash from FCS_COP.1/HMAC*]	[**selection**: *128*, *192*, *256*]bits	(NIST-KDRV [**selection**: *List of Stand in FCS_COP.1/ and FCS_COP.1/*

Application Note: Note that Camellia algorithms do not support 192-bit key sizes. The interface referenced in the requirement could take different forms, the most likely of which is an application programming interface to an OS kernel. There may be various levels of abstraction. For Authorization Factor Submasks, the key size to be used in the HMAC falls into a range between L1 and L2 defined in ISO/IEC 10118 for the appropriate hash function (for example for SHA-256 L1 = 512, L2 =256) where L2 = k = L1.

General note: in order to use a NIST SP 800-108 conformant method of key derivation, the TOE is permitted to implement this with keys as derived as indicated in Key Derivation Functions table above, and with the algorithms as indicated in the same table.

NIST SP 800-131A Rev 1 allows the use of SHA-1 in these use cases.

KeyDrv5, KeyDrv6, and the XOR option in KeyDrv4 will create an “inverted key hierarchy” in which the TSF will combine two or more keys to create a third key. These same KDFs may also use a submask key as input, which could be an authorization factor or derived from a PBKDF. In these cases the ST author must explicitly declare this option and should present a reasonable argument that the entropy of the inputs to the KDFs will result in full entropy of the expected output.

If keys are combined, the ST author shall describe which method of combination is used in order to justify that the effective entropy of each factor is preserved.

The documentation of the product’s encryption key management should be detailed enough that, after reading, the evaluator will thoroughly understand the product’s key management and how it meets the requirements to ensure the keys are adequately protected. This documentation should include an essay and diagrams. This documentation is not required to be part of the TSS; it can be submitted as a separate document and marked as developer proprietary.

SP 800-56C specifies a two-step key derivation procedure that employs an extraction-thenexpansion technique for deriving keying material from a shared secret generated during a key establishment scheme. The Randomness Extraction step as described in Section 5 of SP 800-56C is followed by Key Expansion using the key derivation functions defined in SP 800-108.

This requirement must be claimed by the TOE if at least one of FCS_CKM.1/KEK, FCS_CKM.1/SK, or FCS_COP.1/KeyEnc chooses a selection related to key derivation.

If at least one of KeyDrv4, KeyDrv5, or KeyDrv6 is selected AND password-based key derivation is used to create at least one of the inputs, the selection-based SFR FCS_COP.1/PBKDF must also be claimed.

Evaluation Activities

FCS_CKM_EXT.5:

FCS_COP.1/Hash Cryptographic Operation (Hashing)

FCS_COP.1.1/Hash

The TSF shall perform [*cryptographic hashing*] in accordance with a specified cryptographic algorithm [**selection**: *SHA-1*, *SHA-224*, *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-224*, *SHA3-256*, *SHA3-384*, *SHA3-512*] that meets the following: [**selection**: *ISO/IEC 10118-3:2018*, *FIPS 180-4*].

Application Note: The hash selection should be consistent with the overall strength of the algorithm used for signature generation. For example, the DSC should choose SHA-256 for 2048-bit RSA or ECC with P-256, SHA-384 for 3072-bit RSA, 4096-bit RSA, or ECC with P-384, and SHA-512 for ECC with P-521. The ST author selects the standard based on the algorithms selected.

SHA-1 may be used for the following applications: generating and verifying hash-based message authentication codes (HMACs), key derivation functions (KDFs), and random bit/number generation (In certain cases, SHA-1 may also be used for verifying old digital signatures and time stamps, provided that this is explicitly allowed by the application domain).

Evaluation Activities

FCS_COP.1/Hash:
TSS
The evaluator shall check that the association of the hash function with other TSF cryptographic functions (for example, the digital signature verification function) is documented in the TSS. The evaluator shall also check that the TSS identifies whether the implementation is bit-oriented or byte-oriented.
Guidance
The evaluator checks the AGD documents to determine that any configuration that is required to configure the required hash sizes is present. The evaluator also checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those hash algorithms selected in the ST. There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

SHA-1 and SHA-2 Tests

The tests below are derived from the "The Secure Hash Algorithm Validation System (SHAVS), Updated: May 21, 2014" from the National Institute of Standards and Technology.

The TSF hashing functions can be implemented with one of two orientations. The first is a byte-oriented implementation: this hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). The second is a bit-oriented implementation: this hashes messages of arbitrary length. Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced by a known-good SHA implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Implementation

The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Short Messages Test, Byte-oriented Implementation

The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Bit-oriented Implementation

*The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the ith message is m + 99*i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.*

Selected Long Messages Test, Byte-oriented Implementation

*The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the ith message is m + 8*99*i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.*

Pseudo-randomly Generated Messages Test

The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of SHAVS, section 6.4. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.

SHA-3 Tests

The tests below are derived from the The Secure Hash Algorithm-3 Validation System (SHA3VS), Updated: April 7, 2016, from the National Institute of Standards and Technology.

For each SHA-3-XXX implementation, XXX represents d, the digest length in bits. The capacity, c, is equal to 2d bits. The rate is equal to 1600-c bits.

65 The TSF hashing functions can be implemented with one of two orientations. The first is a bit-oriented mode that hashes messages of arbitrary length. The second is a byte-oriented mode that hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced by a known-good SHA-3 implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of rate+1 short messages. The length of the messages ranges sequentially from 0 to rate bits. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Short Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of rate/8+1 short messages. The length of the messages ranges sequentially from 0 to rate/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Selected Long Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of 100 long messages ranging in size from rate+(rate+1) to rate+(100(rate+1)), incrementing by rate+1. (For example, SHA-3-256 has a rate of 1088 bits. Therefore, 100 messages will be generated with lengths 2177, 3266, ..., 109988 bits.) The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.*

Selected Long Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of 100 messages ranging in size from (rate+(rate+8)) to (rate+100(rate+8)), incrementing by rate+8. (For example, SHA-3-256 has a rate of 1088 bits. Therefore 100 messages will be generated of lengths 2184, 3280, 4376, ..., 110688 bits.) The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.*

Pseudo-randomly Generated Messages Monte Carlo) Test, Byte-oriented Mode

The evaluators supply a seed of d bits (where d is the length of the message digest produced by the hash function to be tested. This seed is used by a pseudorandom function to generate 100,000 message digests. One hundred of the digests (every 1000th digest) are recorded as checkpoints. The TOE then uses the same procedure to generate the same 100,000 message digests and 100 checkpoint values. The evaluators then compare the results generated ensure that the correct result is produced when the messages are generated by the TSF.

FCS_COP.1/HMAC Cryptographic Operation (Keyed Hash)

FCS_COP.1.1/HMAC

The TSF shall perform [*keyed hash message authentication*] in accordance with a specified cryptographic algorithm [**selection**: *HMAC-SHA-1*, *HMAC-SHA-256*, *HMAC-SHA-384*, *HMAC-SHA-512*, *KMAC128*, *KMAC256*] and cryptographic key sizes [**assignment**: *key size (in bits)*] that meet the following: [**selection**: *ISO/IEC 9797-2:2011 Section 7 "MAC Algorithm 2"*, *[NIST-KDV] section 4 "KMAC"*].

Application Note: The HMAC key size falls into a range between L1 and L2 defined in ISO/IEC 10118 for the appropriate hash function (for example for SHA-256 L1 = 512, L2 = 256) where L2 ≤ k ≤ L1.

Evaluation Activities

FCS_COP.1/HMAC:
TSS
The evaluator shall examine the TSS to ensure that it specifies the following values used by the HMAC and KMAC functions: output MAC length used.
Guidance
There are no guidance evaluation activities for this component. There are no KMD evaluation activities for this component.
Tests
The following test requires the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

This test is derived from The Keyed-Hash Message Authentication Code Validation System (HMACVS), updated 6 May 2016.

The evaluator shall provide 15 sets of messages and keys for each selected hash algorithm and hash length/key size/MAC size combination. The evaluator shall have the TSF generate HMAC or KMAC tags for these sets of test data. The evaluator shall verify that the resulting HMAC or KMAC tags match the results from submitting the same inputs to a known-good implementation of the HMAC or KMAC function, having the same characteristics.

FCS_COP.1/KAT Cryptographic Operation (Key Agreement/Transport)

FCS_COP.1.1/KAT

The TSF shall perform [*cryptographic key agreement/transport*] using the supported methods for key agreement/transport defined by the following rows of Table 5: [**selection**: *KAS1*, *KAS2*, *KTS-OAEP*, *RSAES-PKCS1-v1_5*, *ECDH-NIST*, *ECDH-BPC*, *DH*, *Curve25519*, *ECIES*].

Table 5: Supported Methods for Key Agreement/Transport Operation

Identifier	Cryptographic Algorithm	Key Sizes	List of Standards
KAS1	RSA-single party	[**selection**: *2048*, *3072*, *4096*, *6144*, *8192*]bits	NIST SP 800-56Br2 section 8.2
KAS2	RSA-both party	[**selection**: *2048*, *3072*, *4096*, *6144*, *8192*]bits	NIST SP 800-56Br2 section 8.3
KTS-OAEP	RSA	[**selection**: *2048*, *3072*, *4096*, *6144*, *8192*]bits	NIST SP 800-56Br2 section 9
RSAES-PKCS1-v1_5	RSA	[**selection**: *2048*, *3072*, *4096*, *6144*, *8192*]bits	RFC 8017 Section 7.2
ECDH-NIST	ECDH with NIST curves	[**selection**: *256 (P-256)*, *384 (P-384)*, *512 (P-521)*]	NIST SP 800-56Ar3
ECDH-BPC	ECDH with Brainpool curves	[**selection**: *256 (brainpoolP256r1)*, *384 (brainpoolP384r1*, *512 (brainpoolP512r1)*]	RFC 5639 (Section 3)
DH	Diffie-Hellman	[**selection**: *2048*, *3072*, *4096*, *6144*, *8192*]bits	NIST SP 800-56A rev 3, [**selection**: • *RFC 3526 Section [**selection**: 3, 4, 5, 6, 7]*, • *RFC 7919 Appendices [**selection**: A.1, A.2, A.3, A.4, A.5]*]
Curve25519	ECDH	256 bits	RFC 7748
ECIES	ECIES	[**selection**: *256*, *384*, *512*]bits	[**selection**: *ANSI X9.63*, *IEEE 1363a*, *ISO/IEC 18033-2 Part 2*, *SECG SEC1 sec 5.1*]

Application Note: The selections in this SFR should be consistent with the algorithms selected in FCS_CKM.2.

Evaluation Activities

FCS_COP.1/KAT:
TSS
The evaluator shall ensure that the selected RSA and ECDH key agreement/transport schemes correspond to the key generation schemes selected in FCS_CKM.1/AK, and the key establishment schemes selected in FCS_CKM.2 If the ST selects DH, the TSS shall describe how the implementation meets the relevant sections of RFC 3526 (Section 3-7) and RFC 7919 (Appendices A.1-A.5). If the ST selects ECIES, the TSS shall describe the key sizes and algorithms (e.g. elliptic curve point multiplication, ECDH with either NIST or Brainpool curves, AES in a mode permitted by FCS_COP.1/SKC, a SHA-2 hash algorithm permitted by FCS_COP.1/Hash, and a MAC algorithm permitted by FCS_COP.1/HMAC) that are supported for the ECIES implementation.

The evaluator shall ensure that, for each key agreement/transport scheme, the size of the derived keying material is at least the same as the intended strength of the key agreement/transport scheme, and where feasible this should be twice the intended security strength of the key agreement/transport scheme.

Table 2 of NIST SP 800-57 identifies the key strengths for the different algorithms that can be used for the various key agreement/transport schemes.

Guidance
There are no guidance evaluation activities for this component. There are no KMD evaluation

activities for this component.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall verify the implementation of the key generation routines of the supported schemes using the following tests:

If ECDH-NIST or ECDH-BPC is claimed:

SP800-56A Key Agreement Schemes

The evaluator shall verify a TOE's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator shall also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MACdata and the calculation of MACtag.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role-key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of one set of domain parameter values (FFC) or the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the DKM, the corresponding TOE's public keys (static or ephemeral), the MAC tags, and any inputs used in the KDF, such as the Other Information field OI and TOE id fields.

If the TOE does not use a KDF defined in SP 800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret.

The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values.

If key confirmation is supported, the TSF shall perform the above for each implemented approved MAC algorithm.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator shall obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the TOE should be able to recognize. The evaluator generates a set of 24 (FFC) or 30 (ECC) test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator's public keys, the TOE's public/private key pairs, MACTag, and any inputs used in the KDF, such as the other info and TOE id fields.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the other information field OI, the data to be MACed, or the generated MACTag. If the TOE contains the full or partial (only ECC) public key validation, The evaluator shall also individually inject errors in both parties' static public keys, both parties' ephemeral public keys and the TOE's static private key to assure the TOE detects errors in the public key validation function or the partial key validation function (in ECC only). At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

If KAS1, KAS2, KTS-OAEP, or RSAES-PKCS1-v1_5 is claimed:

SP800-56B and PKCS#1 Key Establishment Schemes

If the TOE acts as a sender, the following evaluation activity shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with or without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA public key, the plaintext keying material, any additional input parameters if applicable, the MacKey and MacTag if key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the TOE with the same inputs (in cases where key confirmation is incorporated, the test shall use the MacKey from the test vector instead of the randomly generated MacKey used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector.

If the TOE acts as a receiver, the following evaluation activities shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with our without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTSOAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA private key, the plaintext keying material (KeyData), any additional input parameters if applicable, the MacTag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform the key establishment decryption operation on the TOE and ensure that the outputted plaintext keying material (KeyData) is equivalent to the plain text keying material in the test vector. In cases where key confirmation is incorporated, the evaluator shall perform the key confirmation steps and ensure that the outputted MacTag is equivalent to the MacTag in the test vector.

The evaluator shall ensure that the TSS describes how the TOE handles decryption errors. In accordance with NIST Special Publication 800-56B, the TOE must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations. If KTS-OAEP is supported, the evaluator shall create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.2.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each.

DH:

The evaluator shall verify the correctness of each TSF implementation of each supported Diffie-Hellman group by comparison with a known good implementation.

Curve25519:

The evaluator shall verify a TOE's implementation of the key agreement scheme using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specification. These components include the calculation of the shared secret K and the hash of K.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement role and hash function combination, the tester shall generate 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the shared secret value K, and the hash of K. The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value K and compare the hash generated from this value.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results. To conduct this test, the evaluator generates a set of 30 test vectors consisting of data sets including the evaluator's public keys and the TOE's public/private key pairs.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value K or the hash of K. At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

ECIES:

The evaluator shall verify the correctness of each TSF implementation of each supported use of ECIES by comparison with a known good implementation.

FCS_COP.1/KeyEnc Cryptographic Operation (Key Encryption)

FCS_COP.1.1/KeyEnc

The TSF shall perform [*key encryption and decryption*] using the methods defined in the following rows of Table 6: [**selection**: *SE1*, *AE1*, *SE2*, *XOR*]

Table 6: Supported Methods for Key Encryption Operation

Identifier	Cryptographic Algorithm	Key Sizes	List of Standards
SE1	Symmetric [**selection**: *AES-CCM*, *AES-GCM*, *AES-CBC*, *AES-CTR*, *AES-KWP*, *AESKW*]	[**selection**: *128*, *192*, *256*] bits	See FCS_COP.1/SKC
AE1	Asymmetric KTS-OAEP	[**selection**: *2048*, *3072*] bits	See FCS_COP.1/SKC
SE2	Symmetric [**selection**: *CAM-CBC*, *CAM-CCM*, *CAM-GCM*]	[**selection**: *128*, *256*]	See FCS_COP.1/KAT

		bits	
XOR	Exclusive OR operation	[**selection**: *128*, *192*, *256*] bits	See FCS_CKM_EXT.5

Application Note: A TOE will use this requirement to specify how the Key Encryption Key (KEK) wraps a symmetric encryption key. A TOE will always need this requirement in order to capture the last stage of the key chain in which the Key Encryption Key (KEK) wraps the symmetric encryption key.

If XOR is selected, the selection-based SFR FCS_CKM_EXT.5 must be claimed by the TOE.

Evaluation Activities

FCS_COP.1/KeyEnc:
TSS
The evaluator shall examine the TSS to ensure that it identifies whether the implementation of this cryptographic operation for key encryption (including key lengths and modes) is an implementation that is tested in FCS_COP.1/SKC.

The evaluator shall check that the TSS includes a description of the key wrap functions and shall check that this uses a key wrap algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table.

Guidance
The evaluator checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those key wrap functions selected in the ST. If multiple key access modes are supported, the evaluator shall examine the guidance documentation to determine that the method of choosing a specific mode/key size by the end user is described.

The evaluator shall examine the KMD to ensure that it describes when the key wrapping occurs, that the KMD description is consistent with the description in the TSS, and that for all keys that are wrapped the TOE uses a method as described in the cPP table. No uncertainty should be left over which is the wrapping key and the key to be wrapped and where the wrapping key potentially comes from i.e. is derived from.

If "AES-GCM" or "AES-CCM" is used the evaluator shall examine the KMD to ensure that it describes how the IV is generated and that the same IV is never reused to encrypt different plaintext pairs under the same key. Moreover in the case of GCM, he must ensure that, at each invocation of GCM, the length of the plaintext is at most (2^32)-2 blocks.

Tests
Refer to FCS_COP.1/SKC for the required testing for each symmetric key wrapping method selected from the table and to FCS_COP.1/KAT for the required testing for each asymmetric key wrapping method selected from the table. Each distinct implementation shall be tested separately.

If the implementation of the key encryption operation is the same implementation tested under FCS_COP.1/SKC or FCS_COP.1/KAT, and it has been tested with the same key lengths and modes, then no further testing is required. If key encryption uses a different implementation, (where "different implementation" includes the use of different key lengths or modes), then the evaluator shall additionally test the key encryption implementation using the corresponding tests specified for FCS_COP.1/SKC or FCS_COP.1/KAT.

FCS_COP.1/pbkdf Cryptographic Operation (Password-Based Key Derivation Functions)

FCS_COP.1.1/pbkdf

The TSF shall perform [*password-based key derivation functions*] in accordance with a specified cryptographic algorithm [*HMAC-* [**selection**: *SHA-256*, *SHA-384*, *SHA-512*]], with [**assignment**: *integer number greater than or equal to 1000*] iterations, and output cryptographic key sizes [**selection**: *128*, *192*, *256*]bits that meet the following standard: [*NIST SP 800-132*].

Application Note: The ST must condition a password into a string of bits prior to using it as input to algorithms that form SKs and KEKs. The ST can perform conditioning using one of the identified hash functions or the process described in NIST SP 800-132; the ST author selects the method used. NIST SP 800-132 requires the use of a pseudo-random function (PRF) consisting of HMAC with an approved hash function.

Appendix A of NIST SP 800-132 recommends setting the iteration count in order to increase the computation needed to derive a key from a password and, therefore, increase the workload of performing a dictionary attack.

The TOE must claim this requirement if it claims FCS_CKM.1/SK and selects an algorithm in the PBK row or claims FCS_CKM_EXT.5 and selects at least one of KeyDrv4, KeyDrv5, or KeyDrv6 AND uses password-based key derivation to create at least one of the inputs.

Evaluation Activities

FCS_COP.1/pbkdf:

FCS_COP.1/SigGen Cryptographic Operation (Signature Generation)

FCS_COP.1.1/SigGen

The TSF shall perform [*digital signature generation*] using the supported methods for signature generation defined in the following rows of Table 7 [**selection**: *SigGen1*, *SigGen2*, *SigGen3*, *SigGen4*, *SigGen5*].

Table 7: Supported Methods for Signature Generation Operation

Identifier	Cryptographic Algorithm	Key Sizes	List of Standards
SigGen1	RSASSA-PKCS1-v1_5 using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	[**selection**: *RFC 8017*, *PKCS #1 v2.2 (Section 8.2)*, *FIPS186-4*, *(Section 5.5)*] (RSASSA-PKCS1-v1_5) [**selection**: *ISO10118-3 (Clause 10, 11)*, *FIPS180-4 (Section 6)*] (SHA)
SigGen2	Digital signature scheme 2 using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	ISO9796-2, (Clause 9) (Digital signature scheme 2) [**selection**: *ISO10118-3 (Clause 10, 11)*, *FIPS180-4 (Section 6)*] (SHA)
SigGen3	Digital signature scheme 3 using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	ISO9796-2, (Clause 10) (Digital signature scheme 3) [**selection**: *ISO10118-3 (Clause 10, 11)*, *FIPS180-4 (Section 6)*] (SHA)
SigGen4	RSASSA-PSS using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	[RFC8017, PKCS#1v2.2 (Section 8.1)] (RSASSAPSS) [**selection**: *ISO10118-3 (Clause 10, 11)*, *FIPS180-4 (Section 6)*] (SHA)
SigGen5	ECDSA on [**selection**: *brainpoolP256r1*, *brainpoolP384r1*, *brainpoolP512r1*, *NIST P-256*, *NIST P-384*, *NIST P-521*] using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	[**selection**: • *[**selection**: ISO14888-3, FIPS186-4 (Section 6)] (EDCSA),* • *RFC5639 (Section 3) (Brainpool Curves),* • *FIPS186-4 (Appendix D.1.2) (NIST Curves)*] [**selection**: *ISO10118-3*

			(Clause 10, 11), FIPS180-4 (Section 6)] (SHA)

Evaluation Activities

FCS_COP.1/SigGen:
TSS
The evaluator shall examine the TSS to ensure that all signature generation functions use the approved algorithms and key sizes.
Guidance
There are no AGD evaluation activities for this component. There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

If SigGen1: RSASSA-PKCS1-v1_5or SigGen4: RSASSA-PSS is claimed:

The below test is derived from The 186-4 RSA Validation System (RSA2VS). Updated 8 July 2014, Section 6.3, from the National Institute of Standards and Technology.

To test the implementation of RSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of modulus size and SHA algorithm. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen2: Digital Signature Scheme 2 (DSS2) or SigGen3: Digital Signature Scheme 3 (DSS3):

To test the implementation of DSS2/3 signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of SHA algorithm, hash size and key size. The evaluator them uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen5: ECDSA is claimed:

The below test is derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS). Updated 18 March 2014, Section 6.4, from the National Institute of Standards and Technology.

To test the implementation of ECDSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of curve, SHA algorithm, hash size, and key size. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

FCS_COP.1/SigVer Cryptographic Operation (Signature Verification)

FCS_COP.1.1/SigVer

The TSF shall perform [*digital signature verification*] using the supported methods for signature verification defined in the following rows of Table 8 [**selection**: *SigVer1*, *SigVer2*, *SigVer3*, *SigVer4*, *SigVer5*].

Table 8: Supported Methods for Signature Verification Operation

Identifier	Cryptographic Algorithm	Key Sizes	List of Standards
SigVer1	RSASSA-PKCS1-v1_5 using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	[**selection**: *RFC 8017*, *PKCS #1 v2.2 (Section 8.2)*, *FIPS186-4*, *(Section 5.5)*] (RSASSA-PKCS1-v1_5) [**selection**: *ISO10118-3 (Clause 10, 11)*, *FIPS180-4 (Section 6)*] (SHA)
SigVer2	Digital signature scheme 2 using [**selection**: *SHA-256*, *SHA-384*, *SHA-512*, *SHA3-256*, *SHA3-384*, *SHA3-512*]	[**selection**: *2048 bit*, *3072 bit*]	ISO9796-2, (Clause 9) (Digital signature scheme 2)

			[**selection**: *ISO10118-3 (Clause 10, 11), FIPS180-4 (Section 6)*] (SHA)
SigVer3	Digital signature scheme 3 using [**selection**: *SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512*]	[**selection**: *2048 bit, 3072 bit*]	ISO9796-2, (Clause 10) (Digital signature scheme 3) [**selection**: *ISO10118-3 (Clause 10, 11), FIPS180-4 (Section 6)*] (SHA)
SigVer4	RSASSA-PSS using [**selection**: *SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512*]	[**selection**: *2048 bit, 3072 bit*]	[RFC8017, PKCS#1v2.2 (Section 8.1)] (RSASSAPSS) [**selection**: *ISO10118-3 (Clause 10, 11), FIPS180-4 (Section 6)*] (SHA)
SigVer5	ECDSA on [**selection**: *brainpoolP256r1, brainpoolP384r1, brainpoolP512r1, NIST P-256, NIST P-384, NIST P-521*] using [**selection**: *SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512*]	[**selection**: *2048 bit, 3072 bit*]	[**selection**: • *[**selection**: ISO14888-3, FIPS186-4 (Section 6)] (EDCSA),* • *RFC5639 (Section 3) (Brainpool Curves),* • *FIPS186-4 (Appendix D.1.2) (NIST Curves)*] [**selection**: *ISO10118-3 (Clause 10, 11), FIPS180-4 (Section 6)*] (SHA)

Evaluation Activities

FCS_COP.1/SigVer:
TSS
The evaluator shall check the TSS to ensure that it describes the overall flow of the signature verification. This should at least include identification of the format and general location (e.g., "firmware on the hard drive device" rather than "memory location 0x00007A4B") of the data to be used in verifying the digital signature; how the data received from the operational environment are brought onto the device; and any processing that is performed that is not part of the digital signature algorithm (for instance, checking of certificate revocation lists).
Guidance
There are no AGD evaluation activities for this component. There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

SigVer1: RSASSA-PKCS1-v1_5 and SigVer4: RSASSA-PSS

These tests are derived from The 186-4 RSA Validation System (RSA2VS), updated 8 Jul 2014, Section 6.4.

The FIPS 186-4 RSA Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and three associated key pairs (d, e) for each combination of selected SHA algorithm, modulus size and hash size. Each private key d is used to sign six pseudorandom messages each of 1024 bits. For five of the six messages, the public key (e), message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer5: ECDSA on NIST and Brainpool Curves

These tests are derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), updated 18 Mar 2014, Section 6.5.

The FIPS 186-4 ECC Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and associated key pair (x, y) for each combination of selected curve, SHA algorithm, modulus size, and hash size. Each private key (x) is used to sign 15 pseudorandom messages of 1024 bits. For eight of the fifteen messages, the message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer2: Digital Signature Scheme 2

The following or equivalent steps shall be taken to test the TSF.

For each supported modulus size, underlying hash algorithm, and length of the trailer field (1- or 2-byte), the evaluator shall generate NT sets of recoverable message (M1), non-recoverable message (M2), salt, public key and signature (Σ).

1. *N_T shall be greater than or equal to 20.*
2. *The length of salts shall be selected from its supported length range of salt. The typical length of salt is equal to the output block length of underlying hash algorithm (see 9.2.2 of ISO/IEC 9796-2:2010).*
3. *The length of recoverable messages should be selected by considering modulus size, output block length of underlying hash algorithm, and length of salt (L_S). As described in Annex D of ISO/IEC 9796-2:2010, it is desirable to maximise the length of recoverable message. The following table shows the maximum bit-length of recoverable message that is divisible by 512, for some combinations of modulus size, underlying hash algorithm, and length of salt. None that 2-byte trailer field is assumed in calculating the maximum length of recoverable message*

Maximum length of recoverable message divisible by 512 (bits)	***Modulus size (bits)***	***Underlying hash algorithm (bits)***	***Length of salt L_S (bits)***
1536	*2048*	*SHA-256*	*128*
1024			*256*
1024		*SHA-512*	*128*
1024			*256*
512			*512*
2560	*3072*	*SHA-256*	*128*
2048			*256*
2048		*SHA-512*	*128*
2048			*256*
1536			*512*

4. *The length of non-recoverable messages should be selected by considering the underlying hash algorithm and usages. If the TSF is used for verifying the authenticity of software/firmware updates, the length of non-recoverable messages should be selected greater than or equal to 2048-bit. With this length range, it means that the underlying hash algorithm is also tested for two or more input blocks.*
5. *The evaluator shall select approximately one half of N_T sets and shall alter one of the values (non-recoverable message, public key exponent or signature) in the sets. In altering public key exponent, the evaluator shall alter the public key exponent while keeping the exponent odd. In altering signatures, the following ways should be considered:*
   a. *Altering a signature just by replacing a bit in the bit-string representation of the signature*
   b. *Altering a signature so that the trailer in the message representative cannot be interpreted. This can be achieved by following ways:*
      - *Setting the rightmost four bits of the message representative to the values other than '1100'.*
      - *In the case when 1-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xbc', while keeping the rightmost four bits to '1100'.*
      - *In the case when 2-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xcc', while keeping the rightmost four bits to '1100'.*

c. *In the case when 2-byte trailer is used, altering a signature so that the hash algorithm identifier in the trailer (i.e. the left most byte of the trailer) does not correspond to hash algorithms identified in the SFR. The hash algorithm identifiers are 0x34 for SHA-256 (see Clause 10 of ISO/IEC 10118-3:2018), and 0x35 for SHA-512 (see Clause 11 of ISO/IEC 10118-3:2018).*
d. *Let L_S be the length of salt, altering a signature so that the intermediate bit string D in the message representative is set to all zeroes except for the rightmost L_S bits of D.*
e. *(non-conformant signature length) Altering a signature so that the length of signature Σ is changed to modulus size and the most significant bit of signature Σ is set equal to '1'.*
f. *(non-conformant signature) Altering a signature so that the integer converted from signature Σ is greater than modulus n.*

The evaluator shall supply the NT sets to the TSF and obtain in response a set of NT Verification-Success or Verification-Fail values. When the VerificationSuccess is obtained, the evaluator shall also obtain recovered message (M 1).*

The evaluator shall verify that Verification-Success results correspond to the unaltered sets and Verification-Fail results correspond to the altered sets.

For each recovered message, the evaluator shall compare the recovered message (M1) with the corresponding recoverable message (M 1) in the unaltered sets.*

The test passes only if all the signatures made using unaltered sets result in Verification-Success, each recovered message (M 1) is equal to corresponding M 1 in the unaltered sets, and all the signatures made using altered sets result in Verification-Fail.*

SigVer3: Digital Signature Scheme 3

The evaluator shall perform the test described in SigVer2: Digital Signature Scheme 2 while using a fixed salt for NT sets.

FCS_COP.1/SKC Cryptographic Operation (Symmetric Key Cryptography)

FCS_COP.1.1/SKC

The TSF shall perform [*data encryption/decryption*] using the supported symmetric-key cryptography methods defined in the following rows of Table 9 [**selection**: *AES-CCM*, *AES-GCM*, *AES-CBC*, *AES-CTR*, *XTS-AES*, *AES-KWP*, *AES-KW*, *CAM-CBC*, *CAM-CCM*, *CAM-GCM*, *XTS-CAM*].

Table 9: Supported Methods for Symmetric Key Cryptography Operation

Identifier	Cryptographic Algorithm	Key Sizes	List of Standards
AES-CCM	AES in CCM mode with unpredictable, nonrepeating nonce, minimum size of 64 bits	[**selection**: *128 bits*, *192 bits*, *256 bits*]	ISO 18033-3 (AES) ISO 19772, Clause 8 (CCM) NIST SP800-38C (CCM)
AES-GCM	AES in GCM mode with non-repeating IVs; IV length must be equal to 96 bits; the deterministic IV construction method (SP800-38D, Section 8.2.1) must be used; the MAC length t must be one of the values [**selection**: *96*, *104*, *112*, *120*, *128*]	[**selection**: *128 bits*, *192 bits*, *256 bits*]	ISO 18033-3 (AES) ISO 19772, Clause 11 (GCM) NIST SP800-38D (GCM)
AES-CBC	AES in CBC mode with non-repeating and unpredictable IVs	[**selection**: *128 bits*, *192 bits*, *256 bits*]	ISO 18033-3 (AES) ISO 10116 (CBC) NIST SP800-38A (CBC)
AES-CTR	AES in counter mode with a non-repeating initial counter and with no repeated use of counter values across multiple messages with the same secret key	[**selection**: *128 bits*, *192 bits*, *256 bits*]	ISO 18033-3 (AES) ISO 10116 (CTR) NIST SP800-38A (CTR)
XTS-AES	AES in XTS mode with unique [**selection**: *consecutive non-negative*	[**selection**: *256 bits*,	ISO 18033-3 (AES)

	integers starting at an arbitrary non-negative integer, data unit sequence numbers] tweak values	*512 bits*]	[**selection**: *IEEE 1619, NIST SP800-38E*](XTS)
AES-KWP	KWP based on AES	[**selection**: *128 bits, 192 bits, 256 bits*]	ISO 18033-3 (AES) NIST SP 800-38F, sec. 6.3 (KWP)
AES-KW	KW based on AES	[**selection**: *128 bits, 192 bits, 256 bits*]	ISO 18033-3 (AES) NIST SP 800-38F, sec. 6.2 (KW) ISO/IEC 19772, clause 7 (key wrap)
CAM-CBC	Camellia in CBC mode with non-repeating and unpredictable IVs	[**selection**: *128 bits, 256 bits*]	ISO 18033-3 (Camellia) ISO 10116 (CBC)
CAM-CCM	Camellia in CCM mode with unpredictable, nonrepeating nonce, minimum size of 64 bits	[**selection**: *128 bits, 256 bits*]	ISO 18033-3 (Camellia) ISO 19772, Clause 8 (CCM) NIST SP800-38C (CCM)
CAM-GCM	Camellia in GCM mode with non-repeating IVs; IV length must be equal to 96 bits; the deterministic IV construction method (SP800-38D, Section 8.2.1) must be used; the MAC length t must be one of the values [**selection**: *96, 104, 112, 120, 128*]	[**selection**: *128 bits, 256 bits*]	ISO 18033-3 (Camellia) ISO 19772, Clause 11 (GCM) NIST SP800-38D (GCM)
XTS-CAM	Camellia in XTS mode with unique [**selection**: *consecutive non-negative integers starting at an arbitrary non-negative integer, data unit sequence numbers*] tweak values	[**selection**: *256 bits, 512 bits*]	ISO 18033-3 (Camellia) [**selection**: *IEEE 1619, NIST SP800-38E*](XTS)

Evaluation Activities

FCS_COP.1/SKC:

TSS

The evaluator shall check that the TSS includes a description of encryption functions used for symmetric key encryption. The evaluator should check that this description of the selected encryption function includes the key sizes and modes of operations as specified in the cPP table 9 "Supported Methods for Symmetric Key Cryptography Operation."

The evaluator shall check that the TSS describes the means by which the TOE satisfies constraints on algorithm parameters included in the selections made for 'cryptographic algorithm' and 'list of standards'.

Guidance

If the product supports multiple modes, the evaluator shall examine the vendor's documentation to determine that the method of choosing a specific mode/key size by the end user is described. The evaluator shall examine the KMD to ensure that the points at which symmetric key encryption and decryption occurs are described, and that the complete data path for symmetric key encryption is described. The evaluator checks that this description is consistent with the relevant parts of the TSS.

Assessment of the complete data path for symmetric key encryption includes confirming that the KMD describes the data flow from the device's host interface to the device's non-volatile memory storing the data, and gives information enabling the user data datapath to be distinguished from those situations in which data bypasses the data encryption engine (e.g. read-write operations to an unencrypted Master Boot Record area). The evaluator shall ensure that the documentation of the data path is detailed enough that it thoroughly describes the parts of the TOE that the data passes through (e.g. different memory types, processors and co-processors), its encryption state (i.e. encrypted or unencrypted) in each part, and any places where the data is stored. For example, any caching or buffering of the data should be identified and distinguished from the final destination in non-volatile memory (the latter represents the location from which the host will expect to retrieve the data in future).

If support for AES-CTR is claimed and the counter value source is internal to the TOE, the evaluator shall verify that the KMD describes the internal counter mechanism used to ensure that it provides unique counter block values.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The following tests are conditional based upon the selections made in the SFR. The evaluator shall perform the following test or witness respective tests executed by the developer. The tests must be executed on a platform that is as close as practically possible to the operational platform (but which may be instrumented in terms of, for example, use of a debug mode). Where the test is not carried out on the TOE itself, the test platform shall be identified and the differences between test environment and TOE execution environment shall be described.

Preconditions for testing:

- *Specification of keys as input parameter to the function to be tested*
- *specification of required input parameters such as modes*
- *Specification of user data (plaintext)*
- *Tapping of encrypted user data (ciphertext) directly in the non-volatile memory*

AES-CBC:
For the AES-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CBC implementation.

AES-CBC Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using a key value of all zeros and an IV of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of an all-zeros plaintext using the given key value and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CBC decryption of an all-zeros ciphertext using the given key and an IV of all zeros.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CBC encryption of each plaintext value using a key of each size and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CBC, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

AES-CBC Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a plaintext message of length i blocks, and encrypt the message using AES-CBC. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a

known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for 2 ≤ i ≤ 10. For each test, the evaluator shall supply a key, an IV, and a ciphertext message of length i blocks, and decrypt the message using AES-CBC. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of AES-CBC encryption.

```
# Input: PT, IV, Key
Key[0] = Key
IV[0] = IV
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], IV[i], PT[0]
        for j = 0 to 999 {
                if (j == 0) {
                        CT[j] = AES-CBC-Encrypt(Key[i], IV[i], PT[j])
                        PT[j+1] = IV[i]
                } else {
                        CT[j] = AES-CBC-Encrypt(Key[i], PT[j])
                        PT[j+1] = CT[j-1]
                }
        }
        Output CT[j]
        If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
        If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] ||
        If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
        IV[i+1] = CT[j]
        PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CBC-Encrypt with AES-CBC-Decrypt.

AES-CCM:

These tests are intended to be equivalent to those described in the NIST document, "The CCM Validation System (CCMVS)," updated 9 Jan 2012, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/ccmtestvectors.zip or use values not generated expressly to exercise the AES-CCM implementation.

The evaluator shall test the generation-encryption and decryption-verification functionality of AES-CCM for the following input parameter and tag lengths:

- ***Keys:*** *All supported and selected key sizes (e.g., 128, 192, or 256 bits).*
- ***Associated Data:*** *Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.*
- ***Payload:*** *Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.*
- ***Nonces:*** *All supported nonce lengths (e.g., 8, 9, 10, 11, 12, 13) in bytes.*
- ***Tag:*** *All supported tag lengths (e.g., 4, 6, 8, 10, 12, 14, 16) in bytes.*

The testing for CCM consists of five tests. To determine correctness in each of the below tests, the evaluator shall compare the ciphertext with the result of encryption of the same inputs with a known good implementation.

Variable Associated Data Test: For each supported key size and associated data length, and any supported payload length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Payload Text: For each supported key size and payload length, and any supported associated data length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Nonce Test: For each supported key size and nonce length, and any supported associated data length, payload length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Tag Test: For each supported key size and tag length, and any supported associated data length, payload length, and nonce length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Decryption-Verification Process Test: To test the decryption-verification functionality of AES-CCM, for each combination of supported associated data length, payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five

should pass.

AES-GCM: *These tests are intended to be equivalent to those described in the NIST document, "The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing," rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.*

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmtestvectors.zip, or use values not generated expressly to exercise the AES-GCM implementation.

The evaluator shall test the authenticated encryption functionality of AES-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

- ***Key size in bits:*** *Each selected and supported key size (e.g., 128, 192, or 256 bits).*
- ***Plaintext length in bits:*** *Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.*
- ***AAD length in bits:*** *Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.*
- ***IV length in bits:*** *Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.*
- ***MAC length in bits:*** *Each supported length (e.g., 128, 120, 112, 104, 96).*

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a known-good implementation.

AES-CTR:

For the AES-CTR tests described below, the plaintext and ciphertext values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CTR implementation.

AES-CTR Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of the given plaintext using a key value of all zeros.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CTR decryption of the given ciphertext using a key value of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of an all-zeros plaintext using the given key value.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CTR decryption of an all-zeros ciphertext using the given key.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CTR, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CTR, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CTR encryption of each plaintext value using a key of each size.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CTR, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CTR decrypt each ciphertext value using key of each size consisting of all zeros.

AES-CTR Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key and a plaintext message of length i blocks, and encrypt the message using AES-CTR. The resulting ciphertext

values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for 2 ≤ i ≤ 10. For each test, the evaluator shall supply a key and a ciphertext message of length i blocks, and decrypt the message using AES-CTR. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CTR Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 2-tuples of pseudo-random values for plaintext and keys.

The evaluator shall supply a single 2-tuple of pseudo-random values for each selected key size. This 2-tuple of plaintext and key is provided as input to the below algorithm to generate the remaining 99 2-tuples, and to run each 2-tuple through 1000 iterations of AES-CTR encryption.

```
# Input: PT, Key
Key[0] = Key
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], PT[0]
        for j = 0 to 999 {
                CT[j] = AES-CTR-Encrypt(Key[i], PT[j])
                PT[j+1] = CT[j]
        }
        Output CT[j]
        If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
        If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] ||
        If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] || CT[j])
        PT[0] = CT[j]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 2-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CTR-Encrypt with AES-CTR-Decrypt. 198 Note additional design considerations for this mode are addressed in the KMD requirements.

XTS-AES: *These tests are intended to be equivalent to those described in the NIST document, "The XTS-AES Validation System (XTSVS)," updated 5 Sept 2013, found at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf*

It is not recommended that evaluators use values obtained from static sources such as the XTS-AES test vectors at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSTestVectors.zip or use values not generated expressly to exercise the XTS-AES implementation.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for AES-128) and 512 bit (for AES-256) keys), the evaluator shall provide up to five data lengths:

- *Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.*
- *Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.*
- *The largest data length supported by the implementation, or 2^16 (65536), whichever is larger.*

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-AES encryption. If both kinds of tweak values are supported then each type of tweak value shall be used in half of every 100 sets of input data, for all combinations of key size and data length. The evaluator shall verify that the resulting ciphertext matches the results from submitting the same inputs to a known-good implementation of XTS-AES.

The evaluator shall test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS- AES decrypt.

The evaluator shall check that the full length keys are created by methods that ensure that the two halves are different and independent.

AES-KWP:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KWP (KWP-AE) using the same test as for AES-KW authenticated-encryption with the following change in the five plaintext lengths:

- *Four lengths that are multiples of 8 bits*
- *The largest supported length less than or equal to 4096 bits.*

The evaluator shall test the authenticated-decryption (KWP-AD) functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticatedencryption with AES-KWP authenticated-decryption. For the Authenticated Decryption test, 20 out of the 100 trials per plaintext length have ciphertext values that fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1: (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext with length greater than 64 semi-blocks, 2) plaintext with bit-length not divisible by 8, and 3) plaintext with length 0.

Test 2: (invalid ciphertext length): *Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, and 4) ciphertext with length of one semi-block.*

Test 3: (invalid ICV2): *Test that the implementation detects invalid ICV2 values by encrypting any plaintext value four times using a different value for ICV2 each time as follows: Start with a base ICV2 of 0xA65959A6. For each of the four tests change a different byte of ICV2 to a different value, so that each of the four bytes is changed once. Verify that the implementation of KWP-AD in the TOE outputs FAIL for each test.*

Test 4: (invalid padding length): *Generate one ciphertext using algorithm KWP-AE with substring [len(P)/8]32 of S replaced by each of the following 32-bit values, where len(P) is the length of P in bits and [ ]32 denotes the representation of an integer in 32 bits:*

- *[0]32*
- *[len(P)/8-8]32*
- *[len(P)/8+8]32*
- *[513]32.*

Verify that the implementation of KWP-AD in the TOE outputs FAIL on those inputs.

Test 5: (invalid padding bits):

If the implementation supports plaintext of length not a multiple of 64-bits, then

```
for each PAD length [1..7]
        for each byte in PAD set a zero PAD value;
                replace current byte by a non-zero value and use the resulting plaint
                        input to algorithm KWP-AE to generate ciphertexts;
                verify that the implementation of KWP-AD in the TOE outputs FAIL on
                        this input.
```

AES-KW:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KW for each combination of the following input parameters:

- *Supported key lengths selected in the ST (e.g. 128 bits, 256 bits)*
- *Five plaintext lengths:*
  - *Two lengths that are non-zero multiples of 128 bits (two semi-block lengths)*
  - *Two lengths that are odd multiples of the semi-block length (64 bits)*
  - *The largest supported plaintext length less than or equal to 4096 bits.*

For each set of the above parameters the evaluator shall generate a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator shall compare the results with those obtained from the AES-KW authenticated-encryption function of a known good implementation.

The evaluator shall test the authenticated-decryption functionality of AES-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption (KW-AE) with AES-KW authenticated-decryption (KW-AD). For the authenticated-decryption test, 20 out of the 100 trials per plaintext length must have ciphertext values that are not authentic; that is, they fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1 (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext length greater than 64 semi-blocks, 2) plaintext bit-length not divisible by 64, 3) plaintext with length 0, and 4) plaintext with one semi-block.

Test 2 (invalid ciphertext length):

Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, 4) ciphertext with length of one semiblock, and 5) ciphertext with length of two semi-blocks.

Test 3 (invalid ICV1):

222 Test that the implementation detects invalid ICV1 values by encrypting any plaintext value eight times using a different value for ICV1 each time as follows: Start with a base ICV1 of 0xA6A6A6A6A6A6A6A6. For each of the eight tests change a different byte to a different value, so that each of the eight bytes is changed once. Verify that the implementation of KW-AD in the TOE outputs FAIL for each test.

CAM-CBC:

To test the encrypt and decrypt functionality of Camellia in CBC mode, the evaluator shall perform the tests as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-CCM:

To test the encrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-GCM:

To test the encrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

XTS-CAM:

These tests are intended to be equivalent to those described in the IPA document, ATR-01-B, "Specifications of Cryptographic Algorithm Implementation Testing — Symmetric-Key Cryptography", found at https://www.ipa.go.jp/security/jcmvp/jcmvp_e/documents/atr/atr01b_en.pdf.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for Camellia-128) and 512 bit (for Camellia256) keys), the evaluator shall provide up to five data lengths:

- *Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.*
- *Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.*
- *The largest data length supported by the implementation, or 2^16 (65536), whichever is larger.*

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-Camellia encryption. If both kinds of tweak values are supported, 50 of each 100 sets of input data shall use each type of tweak value. The resulting ciphertext shall be compared to the results of a known-good implementation.

As a prerequisite for this test, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

The evaluator shall test the decrypt functionality of XTS-Camellia using the same test as for encrypt, replacing plaintext values with ciphertext values and XTSCamellia encrypt with XTS-Camellia decrypt.

As a prerequisite for this test, the evaluator shall perform the test for decrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

FCS_RBG_EXT.1 Random Bit Generation

FCS_RBG_EXT.1.1

The TSF shall perform all deterministic random bit generation services in accordance with ISO/IEC 18031:2011 using [**selection**: *Hash_DRBG (any)*, *HMAC_DRBG (any)*, *CTR_DRBG (AES)*].

FCS_RBG_EXT.1.2

The deterministic RBG shall be seeded by at least one entropy source in accordance with NIST SP 800-90B that accumulates entropy from [**selection**: *[**assignment**: number of software-based sources] software-based noise source*, *[**assignment**: number of hardware-based sources] hardware-based noise source*] with a minimum of [**selection**: *128*, *192*, *256*] bits of entropy at least equal to the greatest security strength, according to ISO/IEC 18031:2011, of the keys and CSPs that it will generate.

Application Note: ISO/IEC 18031:2011 contains three different methods of generating random numbers. Each of these in turn depends on underlying cryptographic primitives (hash functions/ciphers). This cPP allows SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 for Hash_DRBG or HMAC_DRBG and only AES-based implementations for CTR_DRBG.

Evaluation Activities

FCS_RBG_EXT.1:

FCS_SLT_EXT.1 Cryptographic Salt Generation

FCS_SLT_EXT.1.1

The TSF shall use salts and nonces generated by an RBG as specified in FCS_RBG_EXT.1.

Evaluation Activities

FCS_SLT_EXT.1:

FCS_STG_EXT.1 Protected Storage

FCS_STG_EXT.1.1

The TSF shall provide [**selection**: *mutable hardware-based*, *immutable hardware-based*, *software-based*] protected storage for asymmetric private keys and [**selection**: *symmetric keys*, *persistent secrets*, *no other keys*].

Application Note: If the protected storage is implemented in software that is protected as required by FCS_STG_EXT.2, the ST author is expected to select "software-based." If "software-based" is selected, the ST author is expected to select all "software-based key storage" in FCS_STG_EXT.2.

Support for protected storage for all symmetric keys and persistent secrets will be required in future revisions.

FCS_STG_EXT.1.2

FCS_STG_EXT.1.2 The TSF shall support the capability of [**selection**: *importing keys/secrets into the TOE*, *causing the TOE to generate keys/secrets*] upon request of [**selection**: *a client application*, *an administrator*].

FCS_STG_EXT.1.3

The TSF shall be capable of destroying keys/secrets in the protected storage upon request of [**selection**: *a client application*, *an administrator*].

FCS_STG_EXT.1.4

The TSF shall have the capability to allow only the user that [**selection**: *imported the key/secret*, *caused the key/secret to be generated*] to use the key/secret. Exceptions may be explicitly authorized only by [**selection**: *the client application*, *the administrator*].

FCS_STG_EXT.1.5

The TSF shall allow only the user that [**selection**: *imported the key/secret*, *caused the key/secret to be generated*] to request that the key/secret be destroyed. Exceptions may only be explicitly authorized by [**selection**: *the client application*, *the administrator*].

Application Note: Not all conformant TOEs will have the ability to import pre-generated keys into the TOE. In these cases, the TOE's ability to receive commands to perform key generation is considered to be its implementation of the Parse service. A subject that caused a key to be generated is considered to be the 'owner' of that key in the same manner as they would be if they had imported it directly.

Evaluation Activities

FCS_STG_EXT.1:

FCS_STG_EXT.2 Key Storage Encryption

FCS_STG_EXT.2.1

The TSF shall encrypt [*AKs, SKs, KEKs, and [**selection**: long-term trusted channel key material, all software-based key storage, no other keys]*] using one of the following methods: [**assignment**: *key encryption methods as specified in FCS_COP.1/KeyEnc*].

Evaluation Activities

FCS_STG_EXT.2:

FCS_STG_EXT.3 Key Integrity Protection

FCS_STG_EXT.3.1

The TSF shall protect the integrity of any encrypted [*AKs, SKs, KEKs, and [**selection**: long-term trusted channel key material, all software-based key storage, no other keys]*] by using [**selection**:

- *Symmetric encryption in [**selection**: AES_CCM, AES_GCM, AES_KW, AES_KWP, CAM_CCM, CAM_GCM] mode in accordance with FCS_COP.1/SKC*,
- *A hash of the stored key in accordance with FCS_COP.1/Hash*,
- *A keyed hash of the stored key in accordance with FCS_COP.1/HMAC*,
- *A digital signature of the stored key in accordance with FCS_COP.1/SigGen using an asymmetric key that is protected in accordance with FCS_STG_EXT.2*,
- *An immediate application of the key for decrypting the protected data followed by a successful verification of the decrypted data with previously known information*

].

FCS_STG_EXT.3.2

The TSF shall verify the integrity of the [**selection**: *hash*, *digital signature*, *MAC*] of the stored key prior to use of the key.

Application Note: This requirement is not applicable to derived keys that are not stored. It is not expected that a single key will be protected from corruption by multiple of these methods; however, a product may use one integrity-protection method for one type of key and a different method for other types of keys. The documentation of the product's encryption key management should be detailed enough that, after reading, the evaluator will thoroughly understand the product's key management and how it meets the requirements to ensure the keys are adequately protected. This documentation should include an essay and diagrams. This documentation is not required to be part of the TSS – it can be submitted as a separate document and marked as developer proprietary.

Evaluation Activities

FCS_STG_EXT.3:

5.1.2 User Data Protection

FDP_ACC.1 Subset Access Control

FDP_ACC.1.1

The TSF shall enforce the [*Access Control SFP*] on [

- *Subjects: S.DSC, S.Admin, S.CA, S.EPS*
- *Objects: OB.P_SDO, OB.T_SDO, OB.AuthData, OB.Pstate, OB.FAACntr, OB.AntiReplay, OB.Context*
- *Operations: OP.Import, OP.Create, OP.Use, OP.Modify, OP.Attest, OP.Store, OP.Export, OP.Destroy*

].

Application Note: The set of operations specified in the assignment can be collectively referred to as "access." Any subsequent use of the term "access" should be interpreted to refer to one or more of these events.

Evaluation Activities

FDP_ACC.1:

FDP_ACF.1 Security Attribute Based Access Control

FDP_ACF.1.1

The TSF shall enforce the [*Access Control SFP*] to objects based on the following: [*subjects (defined in FDP_ACC.1.1) attempt to perform operations (defined in FDP_ACC.1.1) against objects (defined in FDP_ACC.1.1). Subject and object attributes may be used to determine whether the desired operations are permitted.*

The following are the SFP-relevant security attributes that are associated with the subjects and objects defined in FDP_ACC.1.1, as well as any restrictions on the attribute values:

- *S.DSC*
  - *DSC.ID*
- *S.Admin – none*
- *S.CA*
  - *CA.ID*
- *S.EPS*
  - *EPS.ID*
- *OB.P_SDO*
  - *SDO.ID*
  - *SDO.Type*
  - *SDO.AuthData*
  - *SDO.Reauth*
  - *SDO.Conf*
  - *SDO.Export*
  - *SDO.Integrity*
  - *SDO.Bind*
- *OB.T_SDO – same as OB.P_SDO*
- *OB.AuthData – none*
- *OB.Pstate – none*
- *OB.FAACntr – none*
- *OB.AntiReplay – none*
- *OB.Context– none*

].

FDP_ACF.1.2

The TSF shall enforce the following rules to determine if an operation among

controlled subjects and controlled objects is allowed: [

- *Any subject that has been authorized to perform any operation against any OB.P_SDO or OB.T_SDO object can continue to perform this operation if one of the following conditions is true:*
  - *The object's SDO.Reauth attribute has a value of 'none', indicating that reauthorization is not required for subsequent interactions with the SDO*
  - *The object's SDO.Reauth attribute has a value of 'each use', indicating that reauthorization is required for each interaction with the SDO, and the subject has supplied valid authorization data to the TOE*
- *[assignment: rules automatically enforced by the TSF that always prohibit certain subjectobject-operation actions]*
- *[assignment: rules automatically enforced by the TSF that always permit certain subjectobject-operation actions]*
- *[assignment: rules automatically enforced by the TSF that conditionally permit certain subject-object-operation actions based on subject security attributes, object security attributes, or other conditions]*
- *[selection: [assignment: any configurable rules or parameters that can be modified to affect the behavior of the Access Control SFP], no configurable rules]*

].

FDP_ACF.1.3

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

Application Note: The expectation of this SFR is that the reader is given sufficient information to determine, for each object controlled by the TOE, the operations that can be performed on it based on the subject attempting to perform the operation, and whether this is conditional based on attribute values or any other circumstances.

It is expected that many of the subject-object-operation combinations will always be prohibited by the TSF, either because the target object is not externally modifiable or because the subject lacks the ability to perform the operation in question.

The ST author is not expected to create an exhaustive list of subject-object-operation combinations; it is sufficient to list those that are always permitted and those that are conditionally permitted with the expectation that all remaining combinations are prohibited.

FDP_ACF.1.3 and FDP_ACF.1.4 allow the ST author to optionally specify override conditions to resolve otherwise contradictory Access Control SFP rules. For example, the rule "S.Admin may always modify the SDO.Conf attribute of any OB.P_SDO or OB.T_SDO object" may be overridden by a statement in FDP_ACF.1.4 that identifies any particular SDO objects as nonmodifiable regardless of subject authorizations.

The DSC may contain pre-installed SDOs. The DSC will enforce access control for pre-installed SDOs like any other SDO it contains or manages.

Evaluation Activities

FDP_ACF.1:

FCS_ETC_EXT.2 Propagation of SDOs

FCS_ETC_EXT.2.1

The TSF shall propagate only SDO references, wrapped authorization data, and wrapped SDOs such that only [**selection**: *the TSF*, *authorized users*] can access them.

Application Note: The "SDO reference" is a pointer to an object that resides in the TOE; this can be thought of as a token to the object. The "only the TSF can unwrap the data" selection refers to data that is stored outside the TOE boundary (i.e., data that has been propagated).

Evaluation Activities

FCS_ETC_EXT.2:

FDP_FRS_EXT.1 Factory Reset

FDP_FRS_EXT.1.1

The TSF shall permit a factory reset of the TOE upon: [**selection**: *activation by external interface, presentation of [**assignment**: types of authorization data required and reference to their specification], no actions or conditions*].

Application Note: If the DSC provides factory reset and requires an authorization to carry out the operation then the ST author selects either

presentation of... and fills in the authorization data accepted (e.g. a PIN or a cryptographic token based on some specification referenced in the assigned value). If the DSC provides factory reset external to the DSC without requiring authorization then the ST author selects activation by external interface. This selection is intended for use when the device containing the DSC takes responsibility for obtaining and checking the authorization for factory reset.

If any selection other than no actions or conditions is made in FDP_FRS_EXT.1.1, the selectionbased SFR FDP_FRS_EXT.2 must be claimed.

Evaluation Activities

FDP_FRS_EXT.1:

FDP_ITC_EXT.1 Parsing of SDEs

FDP_ITC_EXT.1.1

The TSF shall support importing SDEs using [**selection**: *physically protected channels as specified in FTP_ITP_EXT.1, encrypted data buffers as specified in FTP_ITE_EXT.1, cryptographically protected data channels as specified in FTP_ITC_EXT.1*].

FDP_ITC_EXT.1.2

The TSF shall verify the integrity of the SDE using [**selection**:

- *cryptographic hash as specified in FCS_COP.1/Hash,*
- *keyed hash as specified in FCS_COP.1/HMAC,*
- *integrityproviding encryption algorithm as specified in FCS_COP.1/KeyEnc [**selection**: SE1, SE2],*
- *digital signature as specified in FCS_COP.1/SigVer,*
- *integrity verification supported by FDP_ITC_EXT.1.1*

].

FDP_ITC_EXT.1.3

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

FDP_ITC_EXT.1.4

The TSF shall bind SDEs to security attributes using [**assignment**: *list of ways the TSF generates security attributes and binds them to the SDEs*].

Application Note: The way the TSF checks the integrity of the SDE depends on the method of importation. For example, the encrypted data channel may provide data integrity as part of its service.

When a TSF parses an SDE, it should generate security attributes and create an SDO by binding the security attributes to the SDE.

If physically protected channels as specified in FTP_ITC_EXT.1 is selected, the selection-based SFR FTP_ITP_EXT.1 must be claimed.

If encrypted data buffers as specified in FTP_ITE_EXT.1 is selected, the selection-based SFR FTP_ITE_EXT.1 must be claimed.

If cryptographically protected data channels as specified in FTP_ITC_EXT.1 is selected, the selection-based SFR FTP_ITC_EXT.1 must be claimed.

Evaluation Activities

FDP_ITC_EXT.1:

FDP_ITC_EXT.2 Parsing of SDOs

FDP_ITC_EXT.2.1

The TSF shall support importing SDOs using [**selection**: *physically protected channels as specified in FTP_ITP_EXT.1, encrypted data buffers as specified in FTP_ITE_EXT.1, cryptographically protected data channels as specified in FTP_ITC_EXT.1*].

FDP_ITC_EXT.2.2

The TSF shall verify the integrity of the SDO using [**selection**:

- *cryptographic hash as specified in FCS_COP.1/Hash,*
- *keyed hash as specified in FCS_COP.1/HMAC,*
- *integrityproviding encryption algorithm as specified in FCS_COP.1/KeyEnc [**selection**: SE1, SE2],*
- *digital signature as specified in FCS_COP.1/SigVer,*
- *integrity verification supported by FDP_ITC_EXT.2.1*

].

FDP_ITC_EXT.2.3

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

FDP_ITC_EXT.2.4

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

FDP_ITC_EXT.2.5

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.

Application Note: The way the TSF checks the integrity of the SDO depends on the method of importation. For example, the encrypted data channel may provide data integrity as part of its service.

When a TSF parses an SDO, it should already have a set of security attributes. However, the TSF may modify these attributes, if authorized, to comply with security policies on the TOE.

If physically protected channels as specified in FTP_ITC_EXT.1 is selected, the selection-based SFR FTP_ITP_EXT.1 must be claimed.

If encrypted data buffers as specified in FTP_ITE_EXT.1 is selected, the selection-based SFR FTP_ITE_EXT.1 must be claimed.

If cryptographically protected data channels as specified in FTP_ITC_EXT.1 is selected, the selection-based SFR FTP_ITC_EXT.1 must be claimed.

Evaluation Activities

FDP_ITC_EXT.2:

FDP_MFW_EXT.1 Mutable/Immutable Firmware

FDP_MFW_EXT.1.1

The TSF shall be maintained as [**selection**: *immutable*, *mutable*] firmware.

Application Note: The ST author must include FDP_MFW_EXT.2, FDP_MFW_EXT.3, FPT_FLS.1/FW, and FPT_RPL.1/Rollback if mutable is selected.

Evaluation Activities

FDP_MFW_EXT.1:

FDP_RIP.1 Subset Residual Information Protection

FDP_RIP.1.1

The TSF shall ensure that any previous information content of a resource is made unavailable upon the [*deallocation of the resource from*] the following objects: [

- *SDOs*
- *SDEs*

].

Application Note: When an SDE is a key then it is also subject to the key destruction requirements in FCS_CKM.4, depending on where and how it is stored. This SFR applies to authorization data that are SDEs and security attributes in SDOs.

Evaluation Activities

FDP_RIP.1:

FDP_SDC_EXT.1 Confidentiality of SDEs

FDP_SDC_EXT.1.1

The TSF shall use [**selection**:

- *protected storage*,
- *symmetric encryption using [**selection**: AES-CCM, AES-GCM, AES-CBC, AES-KWP, AES-KW, CAM-CBC, CAM-CCM, CAM-GCM] as specified in FCS_COP.1/SKC*,
- *key wrapping using [**selection**: KAS1, KAS2, KTS-OAEP] as specified in FCS_COP.1/KAT*

] to protect the confidentiality of authorization data and [**assignment**: *list of internally and externally stored SDEs identified in the Confidential SDE List attribute of an SDO*].

FDP_SDC_EXT.1.2

The TSF shall use FCS_CKM.1/KEK to derive or generate the key to encrypt the SDEs.

Application Note: This SFR applies to confidential SDEs, especially secret and private keys, Allowed Random Number Generators' state data, and vendor verification reference data. This SFR also applies to all authorization data appearing in the attribute list under SDO.AuthData as well as any administrator authorization data which may be stored implicitly.

If the TOE stores these parameters outside of its boundary, it must encrypt them according to the cryptographic requirements for key encryption, as required by FDP_ETC_EXT.2.

Vendor pre-installed SDOs includes both objects installed during manufacturing, and those provisioned by the vendor before final release to customer. The administrator and no one else owns and controls these objects.

The confidential-SDE List attribute of the SDO indicates those SDEs that require confidentiality. If SDEs do not require confidentiality, then its omission from this list indicates that confidentiality is not required.

Evaluation Activities

FDP_SDC_EXT.1:

FDP_SDI.2 Stored Data Integrity Monitoring and Action

FDP_SDI.2.1

The TSF shall monitor SDOs and SDEs controlled by the TSF for [*integrity errors*] on all objects, based on the following attributes: [**selection**:

- *[**assignment**: attribute associated with presence in protected storage],*
- *cryptographic hash,*
- *digital signature,*
- *integrity-providing encryption algorithm as specified in FCS_COP.1/KeyEnc [**selection**: SE1, SE2]*

].

FDP_SDI.2.2

Upon detection of a data integrity error, the TSF shall [

- *prohibit the use of the altered data*
- *send notification of the error where applicable*

].

Application Note: This SFR deals with the mechanism that protects the integrity of the SDEs and security attributes within an SDO. This provides the binding data that ensuresthe prevention of unauthorized changes to the SDEs and attributes.

The cryptographic requirements for cryptographic hashes and digital signatures apply here.

No specific requirement is placed here on the nature of the integrity protection data, but the Security Target shall describe this protection measure, and shall identify the iteration of FCS_COP.1/Hash or FCS_COP.1/HMAC that covers any cryptographic algorithm used.

The integrity protection data in FDP_SDI.2.1 is included in the list of attributes identified in FMT_MSA.1, and protects the value of the SDEs and of the SDO security attributes.

When an SDO is parsed, its integrity is checked when it is imported into the TOE.

Evaluation Activities

FDP_SDI.2:

5.1.3 Identification and Authentication

When a platform process requests the ability to create, use, modify, dispose of, etc., an SDE or SDO within the DSC, as a matter of policy, the DSC may expect or request authorization from the platform process, which may include authentication of the requester on whose behalf the platform process is acting. The DSC assumes the requester to be either a person, a process, or a device. The rules on how the requester formats the request will be outside the scope of this cPP. Upon request (or as a matter of an established protocol), the interface (on behalf of the user) presents to the DSC process those authorization values required to authorize execution of the event request. This may include one or more different types of authentication credentials. The DSC validates these items before acting upon the requested event. The validation may simply compare the authorization values to an expected value, or perform a more complex cryptographic protocol to verify the authenticity of the user. After validation, the DSC may then create and subsequently use an authorization value to represent the validation of these authorization values in anticipation of future requests.

Requirements related to the strength, quality, and performance of authorization values supplied to the DSC, such as X.509 certificates and biometric templates, are all outside the scope of the DSC and are expected to be met by the platform, where applicable. The DSC is only expected to enforce quality metrics on any authorization values it generates itself.

FIA_AFL_EXT.1 Authorization Failure Handling

FIA_AFL_EXT.1.1

The TSF shall maintain [**selection**:

- *a unique counter for [**selection**: each SDO, the following SDOs [**assignment**: list of SDOs]],*
- *one global counter covering [**selection**: all SDOs, the following SDOs*

*[**assignment**: list of SDOs]]*

], called the failed authorization attempt counters, that counts of the number of unsuccessful authorization attempts that occur related to authorizing access to these **SDOs**.

FIA_AFL_EXT.1.2

The TSF shall maintain a [**selection**: *static*, *administrator configurable variable*] threshold of the minimal acceptable number of unsuccessful authorization attempts that occur related to authorizing access to these **SDOs**.

FIA_AFL_EXT.1.3

When the failed authorization attempt counters [**selection**: *meets*, *surpasses*] the threshold for unsuccessful authorization attempts, the TSF shall [**selection**:

- *prevent future authorization attempts for a static prescribed amount of time*,
- *prevent future authorization attempts for an administrator configurable amount of time*,
- *collaborative Protection Profile for Dedicated Security Components*,
- *prevent all future authorization attempts indefinitely (i.e., lock), as described by FIA_AFL_EXT.2*,
- *factory reset the TOE wiping out all non-persistent SDOs, as described by FDP_FRS_EXT.2*

] for these **SDOs**.

FIA_AFL_EXT.1.4

The TSF shall increment the failed authorization attempt counter before it verifies the authorization.

Application Note: The product validates the authorization factors prior to determining whether user (administrator or client application) access to the SDE/SDO is permitted. In cases where validation of the authorization factors fails, the product will not allow access to SDE/SDO. The product validates the authorization factors in such a way that it does not allow an attacker to circumvent the other requirements to gain knowledge about the SDE/SDO or other keying material that protects them from inadvertent exposure. It is possible for the TOE to have different rules for the treatment of different SDOs or groups of SDOs. For example, some SDOs may trigger a factory reset in the event of excessive authorization failures while others may only temporarily block future authorization attempts. The ST author should iterate this SFR for each distinct response the TSF can make (as defined by the selections in FIA_AFL_EXT.1.3) and the SDOs whose authorization failures will trigger these responses.

If prevent all future authorization attempts indefinitely (i.e., lock), as described by FIA_AFL_EXT.2 is selected in FIA_AFL_EXT.1.3, the selection-based SFR FIA_AFL_EXT.2 must be claimed.

If factory reset the TOE wiping out all non-persistent SDOs, as described by FDP_FRS_EXT.2 is selected in FIA_AFL_EXT.1.3, the selection-based SFR FDP_FRS_EXT.2 must be claimed.

Evaluation Activities

FIA_AFL_EXT.1:

FIA_SOS.2 TSF Generation of Secrets

FIA_SOS.2.1

The TSF shall provide a mechanism to generate authorization data that meet [*the following quality metrics:*

- *For each authentication attempt, the probability shall be less than one in 1,000,000 that a random attempt will be successful*
- *For multiple attempts to authenticate during a one-minute period, the probability shall be less than one in 100,000 that a series of random attempts will be successful*

.

FIA_SOS.2.2

The TSF shall be able to enforce the use of TSF generated authorization data for [**assignment**: *non-empty list of TSF functions*].

Application Note: This SFR expects the TSF must generate authorization data from a sufficiently large key space to ensure that users cannot employ random guessing as a statistically plausible method of authorizing actions within the TOE, both for a single event and over a session.

Evaluation Activities

FIA_SOS.2:

FIA_UAU.2 User Authentication before Any Action

FIA_UAU.2.1

The TSF shall require each user **and SDO owner** to be successfully authenticated before **authorizing** any ~~other~~ TSF-mediated actions on behalf of that user **or SDO owner**.

Application Note: This SFR goes with FDP_ACF.1, which authorizes access to SDOs (i.e. authorizes operations with or on SDOs). The security policies in FDP_ACF.1 may require authentication of the subjects and owners of the SDOs before the TSF authorizes access to them. An authentication token is critical data bound to a user. Such data, when presented to the TOE and successfully verified by it, authenticates the user. The TOE may use the successful authentication of a user as an authorization to execute an action on its behalf, or to perform a requested operation on or with an SDO.

This requirement specifies the TSF exercise an authentication mechanism from FIA_UAU.5 by which the TOE authenticates the identity of the user requesting the operation and the owner of the SDO which is an object in the operation. Such authentication is necessary to authorize it to operate with the SDOs. A user could present a unique authentication token. The TSF may accept authentication tokens with no further conditioning. The TSF validates the authentication token prior to granting the authorization to perform the requested operation with the SDO. The SDO security attribute SDO.Reauth determines whether or not the TOE may authenticate the user and the SDO owner only once or each time each time it operates with the SDO.

The means of validation may vary based on the type of authentication token.

Evaluation Activities

FIA_UAU.2:

FIA_UAU.5 Multiple Authentication Mechanisms

FIA_UAU.5.1

The TSF shall provide [**selection**: *none, authentication token mechanism, cryptographic signature mechanism, [**assignment**: list of authentication mechanisms]*] to support user authentication.

FIA_UAU.5.2

The TSF shall authenticate any user's claimed identity according to the [**selection**: *all subject users and SDO owners shall successfully authenticate themselves using one of the mechanisms listed in FIA_UAU.5.1, the Prove service shall not accept "none" as an authentication method, [**assignment**: rules describing how each authentication mechanism provides authentication]*]

Application Note: This SFR describes the authentication mechanisms required for any user of any service as a precondition for providing authorization to execute the service. This includes the authentication of the owner of the SDOs of the service.

Evaluation Activities

FIA_UAU.5:

FIA-UAU.6 Re-Authenticating

FIA-UAU.6.1

The TSF shall re-authenticate the user **for access to an SDO** under the conditions:
[

1. Re-authentication and re-authorization by further successful completion of the authentication and authorization methods in FIA_UAU.2, in accordance with the value of the SDO.Reauth attribute of the SDO as follows:
   a. If SDO.Reauth has the value 'each access';
   b. if SDO.Reauth has the value 'policy' and the TSF determines that the TOE satisfies the policy for re-authentication and reauthorization

]

Application Note: The allowed values for the SDO.Reauth attribute of an SDO are defined in FMT_MSA.3 and the SDO Attributes Initialization Table. The rules in FDP_ACF.1.2 and also ensure that the need for re-authorization has been checked before access to an SDO.

An SDO.Reauth value of 'none' indicates that no authentication of the subject user nor of the SDO owners is necessary. It also indicates that no reauthorization for operations using the SDO is necessary.

An SDO.Reauth value of policy indicates that there may be a more complicated set of circumstances that trigger a re-auth (re-authentication of the users and owners as well as reauthorization of the operation). This could be a policy of a time limit for which a user can use an SDO before re-authentication (e.g. 10

minutes or 24 hours). The ST should indicate the policies allowed, and how the TOE evaluates the policies. The ST should also indicate the location of those policies, and how the TOE protects the integrity of those policies.

When the TSF binds a user to access an SDO, this means that the TSF has authenticated the user and that the TSF authorized the user to have the right to exercise one or more of the following actions: generate the SDO, modify the SDO, including its security attributes, use the SDO in a TOE operation, propagate or duplicate the SDO for use by a device external to the DSC, or destroy the SDO. The user may not have exclusive rights to exercise the operations listed.

Policy as represented by the attributes in the SDO dictates whether or not a user must authenticate itself in order to authorize access to the SDO.

It is possible that the attributes of some SDOs should remain unchanged, and that the attributes of other SDOs may be changed by authorized users. If this is the case, then the ST author should iterate this SFR and indicate in the TSS which SDOs apply to each iteration.

Evaluation Activities

FIA-UAU.6:

5.1.4 TOE Security Functional Requirements Rationale

The following rationale provides justification for each security objective for the TOE, showing that the SFRs are suitable to meet and achieve the security objectives:

Table 10: SFR Rationale

OBJECTIVE	ADDRESSED BY	RATIONALE
O.ACCOUNTABILITY	FAU_GEN.1	'cause FAU_GEN.1 is awesome
	FTP_ITC_EXT.1	Cause FTP reasons
O.INTEGRITY	FPT_SBOP_EXT.1	For reasons
	FPT_ASLR_EXT.1	ASLR For reasons
	FPT_TUD_EXT.1	For reasons
	FPT_TUD_EXT.2	For reasons
	FCS_COP.1/HASH	For reasons
	FCS_COP.1/SIGN	For reasons
	FCS_COP.1/KEYHMAC	For reasons
	FPT_ACF_EXT.1	For reasons
	FPT_SRP_EXT.1	For reasons
	FIA_X509_EXT.1	For reasons
	FPT_TST_EXT.1	For reasons
	FTP_ITC_EXT.1	For reasons
	FPT_W^X_EXT.1	For reasons
	FIA_AFL.1	For reasons
	FIA_UAU.5	For reasons
O.MANAGEMENT	FMT_MOF_EXT.1	For reasons
	FMT_SMF_EXT.1	For reasons
	FTA_TAB.1	For reasons
	FTP_TRP.1	For reasons
O.PROTECTED_STORAGE	FCS_STO_EXT.1, FCS_RBG_EXT.1, FCS_COP.1/ENCRYPT, FDP_ACF_EXT.1	Rationale for a big chunk
O.PROTECTED_COMMS	FCS_RBG_EXT.1, FCS_CKM.1, FCS_CKM.2, FCS_CKM_EXT.4, FCS_COP.1/ENCRYPT, FCS_COP.1/HASH, FCS_COP.1/SIGN, FCS_COP.1/HMAC, FDP_IFC_EXT.1, FIA_X509_EXT.1, FIA_X509_EXT.2, FTP_ITC_EXT.1	Rationale for a big chunk

5.2 Security Assurance Requirements

The Security Objectives in Section 4 Security Objectives were constructed to address threats identified in Section 3.1 Threats. The Security Functional Requirements (SFRs) in Section 5.1 Security Functional Requirements are a formal instantiation of the Security Objectives. The PP identifies the Security Assurance Requirements (SARs) to frame the extent to which the evaluator assesses the documentation applicable for the evaluation and performs independent testing.
This section lists the set of SARs from CC part 3 that are required in evaluations against this PP. Individual Assurance Activities o be performed are specified both in Section 5.1 Security Functional Requirements as well as in this section.
The general model for evaluation of OSs against STs written to conform to this PP is as follows:
After the ST has been approved for evaluation, the TSEF will obtain the OS, supporting environmental IT, and the administrative/user guides for the OS. The ITSEF is expected to perform actions mandated by the Common Evaluation Methodology (CEM) for the ASE and ALC SARs. The ITSEF also performs the Assurance Activities contained within Section 5.1 Security Functional Requirements, which are intended to be an interpretation of the other CEM assurance requirements as they apply to the specific technology instantiated in the OS. The Assurance Activities that are captured in Section 5.1 Security Functional Requirements also provide clarification as to what the developer needs to provide to demonstrate the OS is compliant with the PP.

5.2.1 Class ASE: Security Target

As per ASE activities defined in [CEM].

5.2.2 Class ADV: Development

The information about the OS is contained in the guidance documentation available to the end user as well as the TSS portion of the ST. The OS developer must concur with the description of the product that is contained in the TSS as it relates to the functional requirements. The Assurance Activities contained in Section 5.1 Security Functional Requirements should provide the ST authors with sufficient information to determine the appropriate content for the TSS section.

ADV_FSP.1 Basic Functional Specification (ADV_FSP.1)

The functional specification describes the TSFIs. It is not necessary to have a formal or complete specification of these interfaces. Additionally, because OSs conforming to this PP will necessarily have interfaces to the Operational Environment that are not directly invokable by OS users, there is little point specifying that such interfaces be described in and of themselves since only indirect testing of such interfaces may be possible. For this PP, the activities for this family should focus on understanding the interfaces presented in the TSS in response to the functional requirements and the interfaces presented in the AGD documentation. No additional “functional specification” documentation is necessary to satisfy the assurance activities specified. The interfaces that need to be evaluated are characterized through the information needed to perform the assurance activities listed, rather than as an independent, abstract list.

Developer action elements:

ADV_FSP.1.1D
The developer shall provide a functional specification.

Content and presentation elements:

ADV_FSP.1.2C
The developer shall provide a tracing from the functional specification to the SFRs.

Application Note: As indicated in the introduction to this section, the functional specification is comprised of the information contained in the AGD_OPE and AGD_PRE documentation. The developer may reference a website accessible to application developers and the evaluator. The assurance activities in the functional requirements point to evidence that should exist in the documentation and TSS section; since these are directly associated with the SFRs, the tracing in element ADV_FSP.1.2D is implicitly already done and no additional documentation is necessary.

ADV_FSP.1.3C
The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.

ADV_FSP.1.4C
The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.

ADV_FSP.1.5C
The functional specification shall provide rationale for the implicit categorization of interfaces as SFR-non-interfering.

ADV_FSP.1.6C
The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

Evaluator action elements:

ADV_FSP.1.7E
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.1.8E
The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

Evaluation Activities

> *ADV_FSP.1:*
> *There are no specific assurance activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in Section 5.1 Security Functional Requirements, and other activities described for AGD, ATE, and AVA SARs. The requirements on the content of the functional specification information is implicitly assessed by virtue of the other assurance activities being performed; if the evaluator is unable to perform an activity because there is insufficient interface information, then an adequate functional specification has not been provided.*

5.2.3 Class AGD: Guidance Documentation

The guidance documents will be provided with the ST. Guidance must include a description of how the IT personnel verifies that the Operational Environment can fulfill its role for the security functionality. The documentation should be in an informal style and readable by the IT personnel. Guidance must be provided for every operational environment that the product supports as claimed in the ST. This guidance includes instructions to successfully install the TSF in that environment; and Instructions to manage the security of the TSF as a product and as a component of the larger operational environment. Guidance pertaining to particular security functionality is also provided; requirements on such guidance are contained in the assurance activities specified with each requirement.

AGD_OPE.1 Operational User Guidance (AGD_OPE.1)

Developer action elements:

AGD_OPE.1.1D

The developer shall provide operational user guidance.

Application Note: The operational user guidance does not have to be contained in a single document. Guidance to users, administrators and application developers can be spread among documents or web pages. Rather than repeat information here, the developer should review the assurance activities for this component to ascertain the specifics of the guidance that the evaluator will be checking for. This will provide the necessary information for the preparation of acceptable guidance.

Content and presentation elements:

AGD_OPE.1.2C

The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.

Application Note: User and administrator are to be considered in the definition of user role.

AGD_OPE.1.3C

The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the OS in a secure manner.

AGD_OPE.1.4C

The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate.

Application Note: This portion of the operational user guidance should be presented in the form of a checklist that can be quickly executed by IT personnel (or end-users, when necessary) and suitable for use in compliance activities. When possible, this guidance is to be expressed in the eXtensible Configuration Checklist Description Format (XCCDF) to support security automation. Minimally, it should be presented in a structured format which includes a title for each configuration item, instructions for achieving the secure configuration, and any relevant rationale.

AGD_OPE.1.5C

The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the TSF.

AGD_OPE.1.6C

The operational user guidance shall identify all possible modes of operation of the OS (including operation following failure or operational error), their consequences, and implications for maintaining secure operation.

AGD_OPE.1.7C

The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfill the security objectives for the operational environment as described in the ST.

AGD_OPE.1.8C

The operational user guidance shall be clear and reasonable.

Evaluator action elements:

AGD_OPE.1.9E
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluation Activities

AGD_OPE.1:
Some of the contents of the operational guidance are verified by the assurance activities in Section 5.1 Security Functional Requirements and evaluation of the OS according to the [CEM]. The following additional information is also required. If cryptographic functions are provided by the OS, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the OS. It shall provide a warning to the administrator that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the OS. The documentation must describe the process for verifying updates to the OS by verifying a digital signature – this may be done by the OS or the underlying platform. The evaluator will verify that this process includes the following steps: Instructions for obtaining the update itself. This should include instructions for making the update accessible to the OS (e.g., placement in a specific directory). Instructions for initiating the update process, as well as discerning whether the process was successful or unsuccessful. This includes generation of the hash/digital signature. The OS will likely contain security functionality that does not fall in the scope of evaluation under this PP. The operational guidance shall make it clear to an administrator which security functionality is covered by the evaluation activities.

AGD_PRE.1 Preparative Procedures (AGD_PRE.1)

Developer action elements:

AGD_PRE.1.1D
The developer shall provide the OS, including its preparative procedures.

Application Note: As with the operational guidance, the developer should look to the assurance activities to determine the required content with respect to preparative procedures.

Content and presentation elements:

AGD_PRE.1.2C
The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered OS in accordance with the developer's delivery procedures.

AGD_PRE.1.3C
The preparative procedures shall describe all the steps necessary for secure installation of the OS and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the ST.

Evaluator action elements:

AGD_PRE.1.4E
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AGD_PRE.1.5E
The evaluator shall apply the preparative procedures to confirm that the OS can be prepared securely for operation.

Evaluation Activities

AGD_PRE.1:
As indicated in the introduction above, there are significant expectations with respect to the documentation—especially when configuring the operational environment to support OS functional requirements. The evaluator shall check to ensure that the guidance provided for the OS adequately addresses all platforms claimed for the OS in the ST.

5.2.4 Class ALC: Life-cycle Support

At the assurance level provided for OSs conformant to this PP, life-cycle support is limited to end-user-visible aspects of the life-cycle, rather than an examination of the OS vendor’s development and configuration management process. This is not meant to diminish the critical role that a developer’s practices play in contributing to the overall trustworthiness of a product; rather, it is a reflection on the information to be made available for evaluation at this assurance level.

ALC_CMC.1 Labeling of the TOE (ALC_CMC.1)

This component is targeted at identifying the OS such that it can be distinguished from other products or versions from the same vendor and can be easily specified when being procured by an end user.

Developer action elements:

ALC_CMC.1.1D

The developer shall provide the OS and a reference for the OS.

Content and presentation elements:

ALC_CMC.1.2C

The OS shall be labeled with a unique reference.

Application Note: Unique reference information includes:

- OS Name
- OS Version
- OS Description
- Software Identification (SWID) tags, if available

Evaluator action elements:

ALC_CMC.1.3E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluation Activities

ALC_CMC.1:
The evaluator will check the ST to ensure that it contains an identifier (such as a product name/version number) that specifically identifies the version that meets the requirements of the ST. Further, the evaluator will check the AGD guidance and OS samples received for testing to ensure that the version number is consistent with that in the ST. If the vendor maintains a web site advertising the OS, the evaluator will examine the information on the web site to ensure that the information in the ST is sufficient to distinguish the product.

ALC_CMS.1 TOE CM Coverage (ALC_CMS.1)

Given the scope of the OS and its associated evaluation evidence requirements, this component's assurance activities are covered by the assurance activities listed for ALC_CMC.1.

Developer action elements:

ALC_CMS.1.1D

The developer shall provide a configuration list for the OS.

Content and presentation elements:

ALC_CMS.1.2C

The configuration list shall include the following: the OS itself; and the evaluation evidence required by the SARs.

ALC_CMS.1.3C

The configuration list shall uniquely identify the configuration items.

Evaluator action elements:

ALC_CMS.1.4E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluation Activities

ALC_CMS.1:
The "evaluation evidence required by the SARs" in this PP is limited to the information in the ST coupled with the guidance provided to administrators and users under the AGD requirements. By ensuring that the OS is specifically identified and that this identification is consistent in the ST and in the AGD guidance (as done in the assurance activity for ALC_CMC.1), the evaluator implicitly confirms the information required by this component. Life-cycle support is targeted aspects of the developer's life-cycle and instructions to providers of applications for the developer's devices, rather than an in-depth examination of the TSF manufacturer's development and configuration management process. This is not meant to diminish the critical role that a developer's practices play in contributing to the overall trustworthiness of a product; rather, it's a reflection on the information to be made available for evaluation.
The evaluator will ensure that the developer has identified (in guidance documentation for application developers concerning the targeted platform) one or more development environments appropriate for use in developing applications for the developer's platform. For each of these development environments, the developer shall provide information on how to configure the environment to ensure that buffer overflow protection mechanisms in the environment(s) are invoked (e.g., compiler and linker flags). The evaluator will ensure that this documentation also includes an indication of whether such protections are on by default, or have to be specifically enabled. The evaluator will ensure that the TSF is uniquely identified (with respect to other products from the TSF vendor), and that documentation provided by the developer in association with the requirements in the ST is associated with the TSF using this unique identification.

ALC_TSU_EXT.1 Timely Security Updates

This component requires the OS developer, in conjunction with any other necessary parties, to

provide information as to how the end-user devices are updated to address security issues in a timely manner. The documentation describes the process of providing updates to the public from the time a security flaw is reported/discovered, to the time an update is released. This description includes the parties involved (e.g., the developer, carriers(s)) and the steps that are performed (e.g., developer testing, carrier testing), including worst case time periods, before an update is made available to the public.

Developer action elements:

ALC_TSU_EXT.1.1D
The developer shall provide a description in the TSS of how timely security updates are made to the OS.

ALC_TSU_EXT.1.2D
The developer shall provide a description in the TSS of how users are notified when updates change security properties or the configuration of the product.

Content and presentation elements:

ALC_TSU_EXT.1.3C
The description shall include the process for creating and deploying security updates for the OS software.

ALC_TSU_EXT.1.4C
The description shall include the mechanisms publicly available for reporting security issues pertaining to the OS.

Note: The reporting mechanism could include web sites, email addresses, as well as a means to protect the sensitive nature of the report (e.g., public keys that could be used to encrypt the details of a proof-of-concept exploit).

Evaluator action elements:

ALC_TSU_EXT.1.5E
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluation Activities

ALC_TSU_EXT.1:
The evaluator will verify that the TSS contains a description of the timely security update process used by the developer to create and deploy security updates. The evaluator will verify that this description addresses the entire application. The evaluator will also verify that, in addition to the OS developer's process, any third-party processes are also addressed in the description. The evaluator will also verify that each mechanism for deployment of security updates is described.
The evaluator will verify that, for each deployment mechanism described for the update process, the TSS lists a time between public disclosure of a vulnerability and public availability of the security update to the OS patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator will verify that this time is expressed in a number or range of days.
The evaluator will verify that this description includes the publicly available mechanisms (including either an email address or website) for reporting security issues related to the OS.
The evaluator shall verify that the description of this mechanism includes a method for protecting the report either using a public key for encrypting email or a trusted channel for a website.

5.2.5 Class ATE: Tests

Testing is specified for functional aspects of the system as well as aspects that take advantage of design or implementation weaknesses. The former is done through the ATE_IND family, while the latter is through the AVA_VAN family. At the assurance level specified in this PP, testing is based on advertised functionality and interfaces with dependency on the availability of design information. One of the primary outputs of the evaluation process is the test report as specified in the following requirements.

ATE_IND.1 Independent Testing – Conformance (ATE_IND.1)

Testing is performed to confirm the functionality described in the TSS as well as the administrative (including configuration and operational) documentation provided. The focus of the testing is to confirm that the requirements specified in Section 5.1 Security Functional Requirements being met, although some additional testing is specified for SARs in Section 5.2 Security Assurance Requirements. The Assurance Activities identify the additional testing activities associated with these components. The evaluator produces a test report documenting the plan for and results of testing, as well as coverage arguments focused on the platform/OS combinations that are claiming conformance to this PP. Given the scope of the OS and its associated evaluation evidence requirements, this component's assurance activities are covered by the assurance activities listed for ALC_CMC.1.

Developer action elements:

ATE_IND.1.1D
The developer shall provide the OS for testing.

Content and presentation elements:

ATE_IND.1.2C
The OS shall be suitable for testing.

Evaluator action elements:

ATE_IND.1.3E

The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

ATE_IND.1.4E

The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified.

Application Note: The evaluator will test the OS on the most current fully patched version of the platform.

Evaluation Activities

ATE_IND.1:
The evaluator will prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this PP's Assurance Activities.
While it is not necessary to have one test case per test listed in an Assurance Activity, the evaluator must document in the test plan that each applicable testing requirement in the ST is covered. The test plan identifies the platforms to be tested, and for those platforms not included in the test plan but included in the ST, the test plan provides a justification for not testing the platforms. This justification must address the differences between the tested platforms and the untested platforms, and make an argument that the differences do not affect the testing to be performed. It is not sufficient to merely assert that the differences have no affect; rationale must be provided. If all platforms claimed in the ST are tested, then no rationale is necessary. The test plan describes the composition of each platform to be tested, and any setup that is necessary beyond what is contained in the AGD documentation. It should be noted that the evaluator is expected to follow the AGD documentation for installation and setup of each platform either as part of a test or as a standard pre-test condition. This may include special test drivers or tools. For each driver or tool, an argument (not just an assertion) should be provided that the driver or tool will not adversely affect the performance of the functionality by the OS and its platform. This also includes the configuration of the cryptographic engine to be used. The cryptographic algorithms implemented by this engine are those specified by this PP and used by the cryptographic protocols being evaluated (IPsec, TLS). The test plan identifies high-level test objectives as well as the test procedures to be followed to achieve those objectives. These procedures include expected results.
The test report (which could just be an annotated version of the test plan) details the activities that took place when the test procedures were executed, and includes the actual results of the tests. This shall be a cumulative account, so if there was a test run that resulted in a failure; a fix installed; and then a successful re-run of the test, the report would show a "fail" and "pass" result (and the supporting details), and not just the "pass" result.

5.2.6 Class AVA: Vulnerability Assessment

For the first generation of this protection profile, the evaluation lab is expected to survey open sources to discover what vulnerabilities have been discovered in these types of products. In most cases, these vulnerabilities will require sophistication beyond that of a basic attacker. Until penetration tools are created and uniformly distributed to the evaluation labs, the evaluator will not be expected to test for these vulnerabilities in the OS. The labs will be expected to comment on the likelihood of these vulnerabilities given the documentation provided by the vendor. This information will be used in the development of penetration testing tools and for the development of future protection profiles.

AVA_VAN.1 Vulnerability Survey (AVA_VAN.1)

Developer action elements:

AVA_VAN.1.1D

The developer shall provide the OS for testing.

Content and presentation elements:

AVA_VAN.1.2C

The OS shall be suitable for testing.

Evaluator action elements:

AVA_VAN.1.3E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.1.4E

The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the OS.

Application Note: Public domain sources include the Common Vulnerabilities and Exposures (CVE) dictionary for publicly-known vulnerabilities. Public domain sources also include sites which provide free checking of files for viruses.

AVA_VAN.1.5E

The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the OS is resistant to attacks performed by an

attacker possessing Basic attack potential.

Evaluation Activities

AVA_VAN.1:
The evaluator will generate a report to document their findings with respect to this requirement. This report could physically be part of the overall test report mentioned in ATE_IND, or a separate document. The evaluator performs a search of public information to find vulnerabilities that have been found in similar applications with a particular focus on network protocols the application uses and document formats it parses. The evaluator documents the sources consulted and the vulnerabilities found in the report.
For each vulnerability found, the evaluator either provides a rationale with respect to its non-applicability, or the evaluator formulates a test (using the guidelines provided in ATE_IND) to confirm the vulnerability, if suitable. Suitability is determined by assessing the attack vector needed to take advantage of the vulnerability. If exploiting the vulnerability requires expert skills and an electron microscope, for instance, then a test would not be suitable and an appropriate justification would be formulated.

Appendix A - Optional Requirements

As indicated in the introduction to this PP, the baseline requirements (those that must be performed by the TOE) are contained in the body of this PP. This appendix contains three other types of optional requirements that may be included in the ST, but are not required in order to conform to this PP. However, applied modules, packages and/or use cases may refine specific requirements as mandatory.

The first type (A.1 Strictly Optional Requirements) are strictly optional requirements that are independent of the TOE implementing any function. If the TOE fulfills any of these requirements or supports a certain functionality, the vendor is encouraged to include the SFRs in the ST, but are not required in order to conform to this PP.

The second type (A.2 Objective Requirements) are objective requirements that describe security functionality not yet widely available in commercial technology. The requirements are not currently mandated in the body of this PP, but will be included in the baseline requirements in future versions of this PP. Adoption by vendors is encouraged and expected as soon as possible.

The third type (A.3 Implementation-based Requirements) are dependent on the TOE implementing a particular function. If the TOE fulfills any of these requirements, the vendor must either add the related SFR or disable the functionality for the evaluated configuration.

A.1 Strictly Optional Requirements

This PP does not define any Strictly Optional requirements.

A.2 Objective Requirements

This PP does not define any Objective requirements.

A.3 Implementation-based Requirements

This PP does not define any Implementation-based requirements.

Appendix B - Selection-based Requirements

As indicated in the introduction to this PP, the baseline requirements (those that must be performed by the TOE or its underlying platform) are contained in the body of this PP. There are additional requirements based on selections in the body of the PP: if certain selections are made, then additional requirements below must be included.

This PP does not define any Selection-based requirements.

Appendix C - Inherently Satisfied Requirements

This appendix lists requirements that should be considered satisfied by products successfully evaluated against this Protection Profile. However, these requirements are not featured explicitly as SFRs and should not be included in the ST. They are not included as standalone SFRs because it would increase the time, cost, and complexity of evaluation. This approach is permitted by [CC] Part 1, **8.2 Dependencies between components**.
This information benefits systems engineering activities which call for inclusion of particular security controls. Evaluation against the Protection Profile provides evidence that these controls are present and have been evaluated.

Requirement	Rationale for Satisfaction
FIA_UAU.1 - Timing of authentication	FIA_AFL.1 implicitly requires that the OS perform all necessary actions, including those on behalf of the user who has not been authenticated, in order to authenticate; therefore it is duplicative to include these actions as a separate assignment and test.
FIA_UID.1 - Timing of identification	FIA_AFL.1 implicitly requires that the OS perform all necessary actions, including those on behalf of the user who has not been identified, in order to authenticate; therefore it is duplicative to include these actions as a separate assignment and test.
FMT_SMR.1 - Security roles	FMT_MOF_EXT.1 specifies role-based management functions that implicitly defines user and privileged accounts; therefore, it is duplicative to include separate role requirements.
FPT_STM.1 - Reliable time stamps	FAU_GEN.1.2 explicitly requires that the OS associate timestamps with audit records; therefore it is duplicative to include a separate timestamp requirement.
FTA_SSL.1 - TSF-initiated session locking	FMT_MOF_EXT.1 defines requirements for managing session locking; therefore, it is duplicative to include a separate session locking requirement.
FTA_SSL.2 - User-initiated locking	FMT_MOF_EXT.1 defines requirements for user-initiated session locking; therefore, it is duplicative to include a separate session locking requirement.
FAU_STG.1 - Protected audit trail storage	FPT_ACF_EXT.1 defines a requirement to protect audit logs; therefore, it is duplicative to include a separate protection of audit trail requirements.
FAU_GEN.2 - User identity association	FAU_GEN.1.2 explicitly requires that the OS record any user account associated with each event; therefore, it is duplicative to include a separate requirement to associate a user account with each event.
FAU_SAR.1 - Audit review	FPT_ACF_EXT.1.2 requires that audit logs (and other objects) are protected from reading by unprivileged users; therefore, it is duplicative to include a separate requirement to protect only the audit information.

Appendix D - Acronyms

Appendix E - Selection Rules

This rules in this appendix define which combinations of selections are considered valid. An ST is considered conforming only if it satisfies all rules.

Appendix F - Use Case Templates

F.1 Elephant-own device

Appendix G - Acronyms

Acronym	Meaning
AES	Advanced Encryption Standard
API	Application Programming Interface
API	Application Programming Interface
ASLR	Address Space Layout Randomization
Base-PP	Base Protection Profile
CC	Common Criteria
CEM	Common Evaluation Methodology
CESG	Communications-Electronics Security Group
CMC	Certificate Management over CMS
CMS	Cryptographic Message Syntax
CN	Common Names
CRL	Certificate Revocation List
CSA	Computer Security Act
CSP	Critical Security Parameters
DAR	Data At Rest
DEP	Data Execution Prevention
DES	Data Encryption Standard
DHE	Diffie-Hellman Ephemeral
DNS	Domain Name System
DRBG	Deterministic Random Bit Generator
DSS	Digital Signature Standard
DSS	Digital Signature Standard
DT	Date/Time Vector
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
EST	Enrollment over Secure Transport
FIPS	Federal Information Processing Standards
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
ITSEF	Information Technology Security Evaluation Facility
NIAP	National Information Assurance Partnership
NIST	National Institute of Standards and Technology
OCSP	Online Certificate Status Protocol
OE	Operational Environment
OID	Object Identifier
OMB	Office of Management and Budget

OS	Operating System
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
PP	Protection Profile
PP	Protection Profile
PP-Configuration	Protection Profile Configuration
PP-Module	Protection Profile Module
RBG	Random Bit Generator
RFC	Request for Comment
RNG	Random Number Generator
RNGVS	Random Number Generator Validation System
S/MIME	Secure/Multi-purpose Internet Mail Extensions
SAN	Subject Alternative Name
SAR	Security Assurance Requirement
SFR	Security Functional Requirement
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol
ST	Security Target
SWID	Software Identification
TLS	Transport Layer Security
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSFI	TSF Interface
TSS	TOE Summary Specification
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
XCCDF	eXtensible Configuration Checklist Description Format
XOR	Exclusive Or
app	Application

Appendix H - Bibliography

Identifier	Title
[CC]	Common Criteria for Information Technology Security Evaluation - • Part 1: Introduction and General Model, CCMB-2017-04-001, Version 3.1 Revision 5, April 2017. • Part 2: Security Functional Components, CCMB-2017-04-002, Version 3.1 Revision 5, April 2017. • Part 3: Security Assurance Components, CCMB-2017-04-003, Version 3.1 Revision 5, April 2017.
[CEM]	Common Evaluation Methodology for Information Technology Security - Evaluation Methodology, CCMB-2012-09-004, Version 3.1, Revision 4, September 2012.
[CESG]	CESG - End User Devices Security and Configuration Guidance
[CSA]	Computer Security Act of 1987, H.R. 145, June 11, 1987.
[OMB]	Reporting Incidents Involving Personally Identifiable Information and Incorporating the Cost for Security in Agency Information Technology Investments, OMB M-06-19, July 12, 2006.