Protection Profile for General-Purpose Computing Platforms

Version: 1.0
2021-02-17
National Information Assurance Partnership

Revision History

Version	Date	Comment
Round 1	2020-11-09	Started
1.0	2020-11-09	Initial release

Contents

1 Introduction

1.1 Overview

The scope of this Protection Profile (PP) is to describe the security functionality of General-Purpose Computing Platforms in terms of and to define functional and assurance requirements for such products.

A platform is a collection of hardware devices and firmware that provide the functional capabilities and services needed by tenant software. Such components typically include embedded controllers, trusted platform modules, management controllers, host processors, network interface controllers, graphics processing units, flash memory, storage controllers, storage devices, boot firmware, runtime firmware, human interface devices, and a power supply.

1.2 Terms

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

1.2.1 Common Criteria Terms

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

Protection Profile (Base-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.

1.2.2 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	Countermeasures that prevent attackers, even those with physical access, from extracting

Protection	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.
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

1.3 Compliant Targets of Evaluation

A general-purpose computing platform is a hardware device that it capable of hosting more than one different operating system, virtualization system, or bare-metal application. Typical platform implementations include--but are not limited to--servers, PC clients, laptops, and tablets.

Mobile device platforms as defined in the Protection Profile for Mobile Device Fundamentals and network device platforms as defined in the collaborative Protection Profile for Network Devices are out of scope of this PP. Mobile device and network device platforms must be evaluated against the more specific requirements in their respective specialized PPs.

1.3.1 TOE Boundary

Figure 1: General TOE

The TOE comprises the hardware and firmware necessary for the hosting of tenant software. Generally, tenant software is an operating system or virtualization system, but may also be "bare-metal" applications. Tenant software is outside the TOE boundary.

For example, for a PC Client platform, the hardware and firmware responsible for booting the platform and operation of platform devices (such as BIOS, device controller firmware and plaform management firmware would all be included in the TOE. Operating systems and application software would be outside the TOE. The chassis and case could be inside or outside the TOE depending on the use case.

For server-class hardware, any management controller responsible for updating platform firmware is expressly included within the TOE.

1.3.2 TOE Operational Environment

The TOE has no platform since it is itself a platform. But the TOE does have an operational environment. The OE consists of the physical environment in which the TOE operates (e.g., data center, vehicle, outdoors) and any networks to which the TOE may be connected. Different use cases invoke different requirements depending on the operational environment.

1.4 Use Cases

[USE CASE 1] Server-Class Platform

This use case includes the base requirements for server-class hardware in a data center. There are no additional physical protections required because the platform us assumed to be protected by the operational environment.

For a the list of appropriate selections and acceptable assignment values for this configuration, see D.1 Server-Class Platform.

[USE CASE 2] Server-Class Platform, Enhanced

This use case adds physical protections to the base requirements for server-class hardware. Additional physical protections required because the platform us assumed to be minimally protected by the by the operational environment. This use case can also be used for servers in data centers where there are enhanced security requirements.

For a the list of appropriate selections and acceptable assignment values for this configuration, see D.2

Server-Class Platform, Enhanced.

[USE CASE 3] Portable Clients (laptops, tablets)
This use case includes the base requirements for portable clients or end-user devices.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.3 Portable Clients (laptops, tablets).

[USE CASE 4] Portable Clients (laptops, tablets), Enhanced
This use case adds physical protections to the base requirements for portable clients or end-user devices. it is intended for devices that are used in high-assurance scanrios.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.4 Portable Clients (laptops, tablets), Enhanced.

[USE CASE 5] CSfC EUD
EUDs used in accordance with the CSfC Mobile Access Capability Package can include smart phones, tablets, and laptops. This use case covers the basic CSfC requirements for tablet and laptop EUDs.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.5 CSfC EUD.

[USE CASE 6] CSfC Hardened EUD
EUDs used in accordance with the CSfC Mobile Access Capability Package can include smart phones, tablets, and laptops. This use case adds requirements for the protection of the device to the base CSfC EUD requirements.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.6 CSfC Hardened EUD.

[USE CASE 7] Tactical EUD
This use case adds requirements for portable end user computing devices in a tactial environment.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.7 Tactical EUD.

[USE CASE 8] Desktop clients
This use case covers the requirements for non-portable desktop computing devices in a low-threat physical environment.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.8 Desktop clients.

[USE CASE 9] Thin/Zero Clients
Client platforms that run thin client operating systems. It would be great if we could fold this into one or both of the client platform use cases. At the boot firmware level, are they really any different. CSfC has a thin client option.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.9 Thin/Zero Clients.

[USE CASE 10] IoT Devices
IoT devices are field-located devices without human interfaces when in normal operation. In order to qualify for evaluation under this PP, the device must meet the basic criteria for a general-purpose platform.
For a the list of appropriate selections and acceptable assignment values for this configuration, see D.10 IoT Devices.

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 does not claim conformance to any packages.

3 Security Problem Description

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

The platform has three major security responsibilities:

- ensuring the integrity of its own firmware
- ensuring that it is resilient
- providing security services to tenant workloads

These responsibilities manifest as protecting:

- Platform firmware
- Platform firmware updates
- Tenant initialization (boot)

3.1 Threats

T.PHYSICAL
An attacker with physical access might be able to compromise TOE integrity, subvert TOE protections, or access tenant data through hardware attacks such as probing, physical manipulation, fault-injection, side-channel analysis, environmental stress, or activating disabled features or pre-delivery services.

T.SIDE_CHANNEL_LEAKAGE
An attacker running in a tenant context might be able to leverage physical effects caused by the operation of the TOE to derive sensitive information about other tenants or the TOE.

T.PERSISTENCE
An attacker might be able to establish a permanent presence on the TOE in firmware. This could result in permanent compromise of tenant information, as well as TOE updates. This threat does not encompass attacker presence in tenant software, as tenant software is not part of the TOE.

T.UPDATE_COMPROMISE
An attacker may attempt to provide a compromised update of TOE firmware. Such updates can undermine the security functionality of the device if they are unauthorized, unauthenticated, or are improperly validated using non-secure or weak cryptography.

T.SECURITY_FUNCTIONALITY_FAILURE
An attacker could leverage failed or compromised security functionality to access, change, or modify tenant data, TOE data, or other security functionality of the device.

T.TENANT-BASED_ATTACK
An attacker running software as a tenant can attempt to access or modify TOE firmware or functionality. Note that direct tenant attacks against other tenants are not encompassed by this threat as they out of scope.

T.REMOTE_ATTACK
An attacker from off the TOE can attempt to compromise the TOE through a network interface connected to an active TOE component, such as a management subsystem.

T.UNAUTHORIZED_RECONFIGURATION
An attacker might be able to modify the configuration of the TOE and alter its functionality. This might include, activating dormant subsystems, disabling hardware assists, or altering boot-time behaviors.

T.UNAUTHORIZED_PLATFORM_ADMINISTRATOR
An attacker might be able to attain platform administrator status by defeating or bypassing authentication measures.

3.2 Assumptions

A.PHYSICAL_PROTECTION
The TOE is assumed to be physically protected in its operational environment and thus is not subject to physical attacks that could compromise its security or its ability to support the security of tenant workloads.

A.ROT_INTEGRITY
The TOE includes one or more Roots of Trust composed of TOE firmware, hardware, and pre-installed credentials. Roots of Trust are assumed to be free of malicious capabilities as their integrity cannot be verified.

A.TRUSTED_ADMIN
The administrator of the TOE is not careless, willfully negligent or hostile.

3.3 Organizational Security Policies

If the OS is bound to a directory or management server, the configuration of the OS software must be capable of adhering to the enterprise security policies distributed by them.

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.PHYSICAL	O.OBJECTIVE	The threat T.PHYSICAL is countered by O.OBJECTIVE as this provides for integrit transmitted data.
T.SIDE_CHANNEL_LEAKAGE	O.OBJECTIVE	The threat T.SIDE_CHANNEL_LEAKAGE countered by O.OBJECTIVE as this provid integrity of transmitted data.
T.PERSISTENCE	O.OBJECTIVE	The threat T.PERSISTENCE is countered O.OBJECTIVE as this provides for integrit transmitted data.
T.UPDATE_COMPROMISE	O.OBJECTIVE	The threat T.UPDATE_COMPROMISE is c by O.OBJECTIVE as this provides for inte transmitted data.
T.SECURITY_FUNCTIONALITY_FAILURE	O.OBJECTIVE	The threat T.SECURITY_FUNCTIONALITY_FAILURE countered by O.OBJECTIVE as this provid integrity of transmitted data.
T.TENANT-BASED_ATTACK	O.OBJECTIVE	The threat T.TENANT-BASED_ATTACK is countered by O.OBJECTIVE as this provid integrity of transmitted data.
T.REMOTE_ATTACK	O.OBJECTIVE	The threat T.REMOTE_ATTACK is counte O.OBJECTIVE as this provides for integrit transmitted data.
T.UNAUTHORIZED_RECONFIGURATION	O.OBJECTIVE	The threat T.UNAUTHORIZED_RECONFIGURATION countered by O.OBJECTIVE as this provid integrity of transmitted data.
T.UNAUTHORIZED_PLATFORM_ADMINISTRATOR	O.OBJECTIVE	The threat T.UNAUTHORIZED_PLATFORM_ADMINI is countered by O.OBJECTIVE as this prov integrity of transmitted data.
A.PHYSICAL_PROTECTION	OE.PHYSICAL_PROTECTION	The operational environment objective OE.PHYSICAL_PROTECTION is realized t A.PHYSICAL_PROTECTION.
A.ROT_INTEGRITY	OE.ROT_INTEGRITY	The operational environment objective OE.SUPPLY_CHAIN is realized through A.ROT_INTEGRITY.
A.TRUSTED_ADMIN	OE.TRUSTED_ADMIN	The operational environment objective OE.TRUSTED_ADMIN is realized through A.TRUSTED_ADMIN.
	O.MANAGEMENT	The organizational security policy P.ENTI is enforced through the objective O.MANAGEMENT as this objective repres the enterprise and user assert manageme the OS.

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 Security Audit (FAU)

FAU_GEN.1 Audit Data Generation

This is a selection-based component. Its inclusion depends upon selection from FPT_ROT_EXT.2.2, FPT_PHP_EXT.1.2, , FPT_TUD_EXT.2.5, FPT_TUD_EXT.3.4.

FAU_GEN.1.1

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

1. Start-up and shutdown of the audit functions
2. All auditable events for the [not selected] level of audit
3. All administrative actions
4. Start-up, shutdown, and reboot of the platform
5. Specifically defined auditable events in Table 1
6. [selection: Audit records reaching [assignment: integer value less than 100] percentage of audit capacity, [assignment: other auditable events derived from this profile]]
7. [selection: Specifically defined auditable event in Table 2, no additional auditable events]

Application Note: The ST Author should include this SFR in the ST if the TOE generates audit events for integrity verification or boot failures as indicated by the appropriate selections in FPT_ROT_EXT.1.2 and FPR_ROT_EXT.2.2.

Evaluation Activities

FAU_GEN.1:
TSS
The evaluator shall check the TSS and ensure that it lists all of the auditable events and provides a format for audit records. Each audit record format type shall be covered, along with a brief description of each field. The evaluator shall check to make sure that every audit event type mandated by the PP-Configuration is described in the TSS.

Guidance
The evaluator shall also make a determination of the administrative actions that are relevant in the context of this PP-Configuration. The evaluator shall examine the administrative guide and make a determination of which administrative commands, including subcommands, scripts, and configuration files, are related to the configuration (including enabling or disabling) of the mechanisms implemented in the TOE that are necessary to enforce the requirements specified in the PP and PP-Module(s). The evaluator shall document the methodology or approach taken while determining which actions in the administrative guide are security-relevant with respect to this PP-Configuration.

Tests
The evaluator shall test the TOE's ability to correctly generate audit records by having the TOE generate audit records for the events listed and administrative actions. For administrative actions, the evaluator shall test that each action determined by the evaluator above to be security relevant in the context of this PP is auditable. When verifying the test results, the evaluator shall ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record have the proper entries.

Note that the testing here can be accomplished in conjunction with the testing of the security mechanisms directly.

5.1.2 Cryptographic Support (FCS)

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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall examine the TSS to verify that it describes how the TOE generates an asymmetric key based on the methods selected from cPP Table 13: "Supported Methods for Asymmetric Key Generation". The evaluator shall examine the TSS to verify that it describes how the TOE invokes the methods selected in the ST from the same table. The evaluator shall examine the TSS to verify that it identifies the usage for each row identifier (key type, key size, and list of standards) selected in the ST.
Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
If the TOE uses the generated key in a key chain/hierarchy then the evaluator shall confirm that the KMD describes:

- *If AK1 is selected, then the KMD describes which methods for generating p and q are used*
- *How the key is used as part of the key chain/hierarchy.*

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.

AK1: RSA Key Generation
The below tests are derived from The 186-4 RSA Validation System (RSA2VS), Updated 8 July 2014, Section 6.2, from the National Institute of Standards and Technology.

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d.

FIPS 186-4 Key Pair generation specifies 5 methods for generating the primes p and q.

These are:

1. *Random Primes:*
   - *Provable primes*
   - *Probable primes*
2. *Primes with Conditions:*
   - *Primes p1, p2, q1, q2, p and q shall all be provable primes.*
   - *Primes p1, p2, q1, and q2 shall be provable primes and p and q shall be probable primes*
   - *Primes p1, p2, q1, q2, p and q shall all be probable primes.*

To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair.

For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated by a known good implementation using the same input parameters.

If the TOE generates Random Probable Primes then if possible, the Random Probable primes method should also be verified against a known good implementation as described above. If verification against a known good implementation is not possible, the evaluator shall have the TSF generate 25 key pairs for each supported key length nlen and verify that all of the following are true:

- *n = p*q*
- *p and q are probably prime according to Miller-Rabin tests with error probability <2^(-125)*
- *2^16 < e < 2^256 and e is an odd integer*

- *GCD(p-1,e) = 1*
- *GCD(q-1,e) = 1*
- *|p-q| > 2^(nlen/2 - 100)*
- *p >= squareroot(2)*(2^(nlen/2 -1))*
- *q >= squareroot(2)*(2^(nlen/2 -1))*
- *2^(nlen/2) < d < LCM(p-1,q-1)*
- *e*d = 1 mod LCM(p-1,q-1)*

AK2 & AK3: ECC Key Generation with 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.

ECC Key Generation Test

For each selected curve, and for each key pair generation method as described in FIPS 186-4, section B.4, the evaluator shall require the implementation under test to generate 10 private/public key pairs (d, Q). The private key, d, shall be generated using a random bit generator as specified in FCS_RBG_EXT.1. The private key, d, is used to compute the public key, Q'. The evaluator shall confirm that 0<d<n (where n is the order of the group), and the computed value Q' is then compared to the generated public/private key pairs' public key, Q, to confirm that Q is equal to Q'.

Public Key Validation (PKV) Test

For each supported curve, the evaluator shall generate 12 private/public key pairs using the key generation function of a known good implementation and modify six of the public key values so that they are incorrect, leaving six values unchanged (i.e., correct). To determine correctness, the evaluator shall submit the 12 key pairs to the public key validation (PKV) function of the TOE and shall confirm that the results correspond as expected to the modified and unmodified values.

AK4: DSA Key Generation using Finite-Field Cryptography (FFC)

The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y.

The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p:

- *Primes q and p shall both be provable primes*
- *Primes q and field prime p shall both be probable primes*

and two ways to generate the cryptographic group generator g:

- *Generator g constructed through a verifiable process*
- *Generator g constructed through an unverifiable process.*

The Key generation specifies 2 ways to generate the private key x:

- *len(q) bit output of RBG where 1 ≤ x ≤ q-1*
- *len(q) + 64 bit output of RBG, followed by a mod q-1 operation and a +1 operation, where 1≤ x≤q-1.*

The security strength of the RBG must be at least that of the security offered by the FFC parameter set.

To test the cryptographic and field prime generation method for the provable primes method or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set.

For each key length supported, the evaluator shall have the TSF generate 25 parameter sets and key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm

- *g != 0,1*
- *q divides p-1*
- *g^q mod p = 1*
- *g^x mod p = y*

for each FFC parameter set and key pair.

AK5: Curve25519 Key Generation

The evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated as specified in RFC 7748 using an approved random bit generator (RBG) and shall be written in littleendian order (least significant byte first. To determine correctness, the evaluator shall submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

Note: Assuming the PKV function of the good implementation will (using little-endian order):

- *Confirm the private and public keys are 32-byte values*
- *Confirm the three least significant bits of the first byte of the private key are zero*
- *Confirm the most significant bit of the last byte is zero*
- *Confirm the second most significant bit of the last byte is one*
- *Calculate the expected public key from the private key and confirm it matches the supplied public key*

The evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify 5 of the public key values so that they are incorrect, leaving five values unchanged (i.e. correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

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

This is an optional component. However, applied modules or packages might redefine it

as mandatory.

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 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:
TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains an SK through direct generation as specified in FCS_RBG_EXT.1, FCS_COP.1/KDF, or FCS_COP.1/PBKDF. The evaluator shall review the TSS to verify that it describes how the ST invokes the functionality described by FCS_RBG_EXT.1 and FCS_COP.1/PBKDF where applicable.

[conditional] If the symmetric key is generated by an RBG, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1 or documentation available for the operational environment 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.
Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
The evaluator shall confirm that the KMD describes, as applicable:

- *The RBG interface and how the ST uses it in symmetric key generation*
- *The KDF interface and how the ST uses it in symmetric key generation*
- *The PBKDF interface and how the ST uses it in symmetric key generation*
- *If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the ST uses the key as part of the key chain/hierarchy.*

Tests
For each selected key generation method, the evaluator shall configure the selected generation

capability. The evaluator shall use the description of the RBG interface to verify that the TOE requests and receives an amount of RBG output greater than or equal to the requested key size. The evaluator shall perform the tests as described for FCS_COP.1/KDF and FCS_COP.1/PBKDF.

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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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.
KMD
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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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.
KMD
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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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: The platform 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.

KMD
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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

FCS_CKM_EXT.4.1

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

Application Note: The platform 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.
KMD
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

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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 Standa
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 800-1((Section 5.1) (K in Counter Mod [**selection**: *ISC CMAC*, *NIST-CMAC*, *ISO-CIF ISO-HMAC*, *FIF HMAC*, *ISO-HA 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 800-1((Section 5.2) (K in Feedback Mo [**selection**: *ISC CMAC, NIST-CMAC, ISO-CIF ISO-HMAC, FIF HMAC, ISO-HA 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 800-1((Section 5.3) (K in n Double Pipeline Iterati Mode) [**selection**: *ISC CMAC, NIST-CMAC, ISO-CIF ISO-HMAC, FIF HMAC, ISO-HA 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**: *ISC HASH, FIPSSH*
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 800-1([selection: (Sec 5.1) (KDF in Counter Mode) (Section 5.2) (K in Feedback Mode); (Section 5.3) (K in Double-Pipel Iteration Mode) [**selection**: *ISC CMAC, NIST-CMAC, ISO-CIF ISO-HMAC, FIF HMAC, ISO-HA FIPS-SHA*]
KeyDrv6	[**selection**: *symmetric key, initialization vector, authentication token, authorization value, HMAC key, KMAC key*]	Two keys	[**selection**: *AES-CCM, AES-GCM, AES-CBC, AES-KWP, AES-KW, CAM-CBC, CAM-CCM, CAM-GCM*] from FCS_COP.1/SKC Symmetric Key table	[**selection**: *128, 192, 256*]bits	[selection: see I of Standards in FCS_COP.1/SK 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) s [**selection**: *see List of Standar in FCS_COP.1/Ha and FCS_COP.1/HM*
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) s [**selection**: *see List of Standar in FCS_COP.1/Ha and FCS_COP.1/HM*

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:
TSS
The evaluator shall check that the TSS includes a description of the key derivation functions and shall check that this uses a key derivation algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table per row. The evaluator shall confirm that the TSS supports the selected methods.

If KeyDrv5 is selected, the evaluator shall verify that the TSS shows that the total length of the concatenated keys used as input to the KDF is greater than or equal to the length of the output from the KDF.

[conditional] If key combination is used to form a KEK, the evaluator shall verify that the TSS describes the method of combination and that this method is either an XOR, a KDF, or encryption.

[conditional] If a KDF is used to form a KEK, the evaluator shall ensure that the TSS includes a description of the key derivation function and shall verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP 800-108.

[conditional] If key concatenation is used to derive KEKs (KeyDrv5), the evaluator shall ensure the TSS includes a description of the randomness extraction step, including the following:

- *The description must include how an approved untruncated MAC function is being used for the randomness extraction step and the evaluator must verify the TSS describes that the output length (in bits) of the MAC function is at least as large as the targeted security strength (in bits) of the parameter set employed by the key establishment scheme (see Tables 1-3 of SP 800-56C).*
- *The description must include how the MAC function being used for the randomness extraction step is related to the PRF used in the key expansion and verify the TSS description includes the correct MAC function:*
  - *If an HMAC-hash is used in the randomness extraction step, then the same HMAC-hash (with the same hash function hash) is used as the PRF in the key expansion step.*
  - *If an AES-CMAC (with key length 128, 192, or 256 bits) is used in the randomness extraction step, then AES-CMAC with a 128-bit key is used as the PRF in the key expansion step.*
- *The description must include the lengths of the salt values being used in the randomness extraction step and the evaluator shall verify the TSS description includes correct salt lengths:*
  - *If an HMAC-hash is being used as the MAC, the salt length can be any value up to the maximum bit length permitted for input to the hash function hash.*
  - *If an AES-CMAC is being used as the MAC, the salt length shall be the same length as the AES key (i.e. 128, 192, or 256 bits).*

Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
The evaluator shall examine the KMD to ensure that:

- *The KMD describes the complete key derivation chain and the description must be consistent with the description in the TSS. For all key derivations the TOE must use a method as described in the cPP table. There should be no uncertainty about how a key is derived from another in the chain.*
- *The length of the key derivation key is defined by the PRF. The evaluator should check whether the key derivation key length is consistent with the length provided by the selected PRF.*
- *If a key is used as an input to several KDFs, each invocation must use a distinct context string. If the output of a KDF execution is used for multiple cryptographic keys, those keys must be disjoint segments of the output.*

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 one or more of the following tests to verify the correctness of the key derivation function, depending on the specific functions that are supported:

Preconditions for testing:
- *Specification of input parameter to the key derivation function to be tested*
- *Specification of further required input parameters*
- *Access to derived keys*

The following table maps the data fields in the tests below to the notations used in SP 800-108 and SP 800-56C

Data Fields	***Notations***	
	SP 800-108	*SP 800-56C*
Pseudorandom function	*PRF*	*PRF*
Counter length	*r*	*r*
Length of output of PRF	*r*	*r*
Length of derived keying material	*L*	*L*
Length of input values	*I_length*	*I_length*
Pseudorandom input values I	*K1 (key derivation key)*	*Z (shared secret)*
Pseudorandom salt values		*S*
Randomness extraction MAC	*n/a*	*MAC*

The below tests are derived from Key Derivation using Pseudorandom Functions (SP 800-108) Validation System (KBKDFVS), Updated 4 January 2016, Section 6.2, from the National Institute of Standards and Technology.

KeyDrv1: Counter Mode Tests:
The evaluator shall determine the following characteristics of the key derivation function:
- *One or more pseudorandom functions that are supported by the implementation (PRF).*
- *One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).*
- *The length (in bits) of the output of the PRF (h).*
- *Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.*
- *Up to two values of L that are NOT evenly divisible by h.*
- *Location of the counter relative to fixed input data: before, after, or in the middle.*
    - *Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.*
    - *Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.*
    - *Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.*
- *The length (I_length) of the input values I.*

For each supported combination of I_length, MAC, salt, PRF, counter location, value of r, and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it. For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output.

The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv2: Feedback Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:
- *One or more pseudorandom functions that are supported by the implementation (PRF).*
- *The length (in bits) of the output of the PRF (h).*
- *Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.*
- *Up to two values of L that are NOT evenly divisible by h.*
- *Whether or not zero-length IVs are supported.*
- *Whether or not a counter is used, and if so:*
    - *One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).*
    - *Location of the counter relative to fixed input data: before, after, or in the middle.*
        - *Counter before fixed input data: fixed input data string length (in bytes), fixed*

input data string value.
    - *Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.*
    - *Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.*
- *The length (I_length) of the input values L.*

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I and pseudorandom salt values. If the KDF supports zero-length IVs, five of these test vectors will be accompanied by pseudorandom IVs and the other five will use zerolength IVs. If zero-length IVs are not supported, each test vector will be accompanied by an pseudorandom IV. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv3: Double Pipeline Iteration Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:

- *One or more pseudorandom functions that are supported by the implementation (PRF).*
- *The length (in bits) of the output of the PRF (h).*
- *Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.*
- *Up to two values of L that are NOT evenly divisible by h.*
- *Whether or not a counter is used, and if so:*
    - *One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).*
    - *Location of the counter relative to fixed input data: before, after, or in the middle.*
        - *Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.*
        - *Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.*
        - *Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.*
- *The length (I_length) of the input values I.*

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv4: Intermediate Keys Method

If the selected algorithm is a hash then the testing of the hash primitive is the only required Evaluation Activity. If the selected algorithm is XOR then no separate primitive testing is necessary.

KeyDrv5: Concatenated Keys Method

The evaluator should confirm that the combined length of the concatenated keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row performed for KeyDrv1, KeyDrv2, and KeyDrv3.

KeyDrv6: Two Keys Method

The evaluator should confirm that the combined length of the two keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row from FCD_COP.1/SK.

KeyDrv7: Shared Secret, Salt, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z) [selection: 128, 256] bits, length of salt (S) [selection: length of input block of PRF, one-half length of input block of PRF, 0] bits, output length (L) [selection: 128, 256] bits, and length of fixed information (FixedInfo) [selection: length of on input block of PRF, onehalf length of input block of PRF, 0] bits, the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit) and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv8: Shared Secret, Salt, IV, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z), length of salt, length of initialization vector (IV), output length (L), and length of fixed information (FixedInfo), the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit), IV (for non-zero lengths, otherwise, use a vector

of length equal to length of input block of PRF and fill with zeros), and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

FCS_COP.1/Hash Cryptographic Operation (Hashing)

This is a selection-based component. Its inclusion depends upon selection from .

FCS_COP.1.1/Hash

The TSF shall perform [*cryptographic hashing*] in accordance with a specified cryptographic algorithm [**selection**: *SHA-1*, *SHA-256*, *SHA-384*, *SHA-512*, *SHA-3-224*, *SHA-3-256*, *SHA-3-384*, *SHA-3-512*] that meet 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 ST Author 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.

Guidance
The evaluator checks the AGD documents to determine that any configuration that is required to be done to configure the functionality for the required hash sizes is present.

Tests
SHA-1 and SHA-2 Tests
The TSF hashing functions can be implemented in one of two modes. The first mode is the byte-oriented mode. In this mode the TSF only 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 mode is the bit-oriented mode. In this mode the TSF hashes messages of arbitrary length. As there are different tests for each mode, an indication is given in the following sections for the bit-oriented vs. the byte-oriented testmacs.

The evaluator shall perform all of the following tests for each hash algorithm implemented by the TSF and used to satisfy the requirements of this PP.

Assurance Activity Note:
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.

Short Messages Test Bit-oriented Mode
The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m 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.

Short Messages Test Byte-oriented Mode
The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range 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 Mode
*The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 99*i, where 1 <= i <= m. 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 m/8 messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 8*99*i, where 1 <= i <= m/8. 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.*

Pseudo-randomly Generated Messages Test
This test is for byte-oriented implementations only. 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]. 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.

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 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.*

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)

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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.
KMD
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)

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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.
KMD
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)

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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*] bits	See FCS_COP.1/KAT
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.

KMD
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)

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall review the TSS to verify that it contains a description of the PBKDF. The evaluator shall also confirm the ST supports the selected hash function itself. The evaluator shall confirm that the TSS contains a description of how the TOE ensures that the output of the PBKDF is at least the same length as that specified in FCS_CKM.1/SK and for the KeyDrv4, KeyDrv5, or KeyDrv6 in FCS_CKM_EXT.5.

If the ST performs additional conditioning, whitening, or manipulation of the password or passphrase before applying the PBKDF, or to the output of the PBKDF, the evaluator shall ensure that the TSS describes the actions and provides assurance that the TSF does not negatively impact the entropy of the PBKDF output.

If any manipulation of the key is performed in forming the submask that will be used to form the KEK, that process shall be described in the TSS.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
No explicit testing of the formation of the submask from the input password is required.

For the NIST SP 800-132-based conditioning of the passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1/HMAC).

The evaluator shall verify that the iteration count for PBKDFs performed by the TOE comply with NIST SP 800-132 by ensuring that the TSS contains a description of the estimated time required to derive key material from passwords and how the TOE increases the computation time for password-based key derivation (including but not limited to increasing the iteration count).

FCS_COP.1/SigGen Cryptographic Operation (Signature Generation)

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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*,	[**selection**: *2048 bit*,	ISO9796-2, (Clause 10)

	SHA-512, *SHA3-256*, *SHA3-384*, *SHA3-512*]	*3072 bit*]	(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.
KMD
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)

This is a selection-based component. Its inclusion depends upon selection from .

FCS_COP.1.1/SigVer

Refinement: The TSF shall perform 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)

Application Note: The ST author should choose the algorithm implemented to perform vertification of digital signatures. if more than one algorithm is available, this requirement should be iterated to specify the functionality. For the algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm. In particular, if ECDSA is selected as one of the signature algorithms, the key size specified must match the selection for the curve used in the algorithm.

For elliptic curve-based schemes, the key size refers to the binary logarithm (log2) of the order of the base point. As the preferred approach for digital signatures, elliptic curves will be required after all the necessary standards and other supporting information are fully established.

If cryptographic signature vertification services are provided to the TOE or to tenant software by a local DSC, then the ST should include an instance of this SFR with instance identifier "(DSC)."

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.
KMD
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)

This is an optional component. However, applied modules or packages might redefine it

as mandatory.

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 integers starting at an arbitrary non-negative integer*, *data unit sequence numbers*] tweak values	[**selection**: *256 bits*, *512 bits*]	ISO 18033-3 (AES) [**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,	[**selection**: *128 bits*,	ISO 18033-3

	minimum size of 64 bits	*256 bits*]	(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.
KMD
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 ≤ i ≤ 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] || CT
        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 $2 \leq i \leq 10$. 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] || CT
        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 plaintext
                        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_ENT_EXT.1 Entropy for Tenant Software

This is an optional component. However, applied modules or packages might redefine it as mandatory.

FCS_ENT_EXT.1.1

The TSF shall provide one or more mechanisms to make entropy suitable for use in FCS_RBG_EXT.1 available to tenant software.

Application Note: This requirement ensures that the TOE makes available sufficient entropy to any tenant that requires it. Every entropy source need not provide high-quality entropy, but tenant software must have a means of acquiring sufficient entropy.

A hardware noise source is a component that produces data that cannot be explained by a deterministic rule, due to its physical nature. In other words, a hardware based noise source generates sequences of random numbers from a physical process that cannot be predicted. For example, a sampled ring oscillator consists of an odd number of inverter gates chained into a loop, with an electrical pulse traveling from inverter to inverter around the loop. The inverters are not clocked, so the precise time required for a complete circuit around the loop varies slightly as various physical effects modify the small delay time at each inverter on the line to the next inverter. This variance results in an approximate natural frequency that contains drift and jitter over time. The output of the ring oscillator consists of the oscillating binary value sampled at a constant rate from one of the inverters – a rate that is significantly slower than the oscillator's natural frequency.

Evaluation Activities

FCS_ENT_EXT.1:
TSS
The evaluator shall verify that the TSS documents the entropy sources implemented by the TOE. It is not necessary to document all the platform features that can be used by tenant software to contribute to entropy, rather only those features expressly provided as entropy sources.
Guidance
The evaluator shall examine the AGD to ensure that it describes how to configure entropy sources (if applicable) and how tenant software can access the sources.
Tests
The evaluator shall perform the following tests:

- ***Test 1:*** *The evaluator shall invoke the entropy source(s) from tenant software. The evaluator shall verify that the tenant acquires values from the interface.*

FCS_RBG_EXT.1 Random Bit Generation

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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: For the selection in FCS_RBG_EXT.1.2, the ST author selects the appropriate number of bits of entropy that corresponds to the greatest security strength of the algorithms included in the ST. Security strength is defined in Tables 2 and 3 of NIST SP 800-57A. For example, if the implementation includes 2048-bit RSA (security strength of 112 bits), AES 128 (security strength 128 bits), and HMAC-SHA-256 (security strength 256 bits), then the ST author would select 256 bits.

FCS_RBG_EXT.1.3

The TSF shall be capable of providing output of the RBG to applications running on the TSF that request random bits.

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.

This requirement must be included in the ST only if the TOE generates keys for its own use, or if it provides RNG services for tenant software.

Evaluation Activities

FCS_RBG_EXT.1:
TSS
The evaluator shall examine the TSS to determine that it specifies the DRBG type, identifies the entropy sources seeding the DRBG, and state the assumed or calculated min-entropy supplied either separately by each source or the min-entropy contained in the combined seed value.

In addition to the materials below, documentation shall be produced—and the evaluator shall perform the activities—in accordance with Appendix D of [DSCcPP].

Guidance
There are no AGD evaluation activities for this component.
KMD
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 perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration.

If the RNG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP800-90A).

If the RNG does not have prediction resistance, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) reseed, (4) generate a second block of random bits (5) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call.

The following paragraphs contain more information on some of the input values to be generated/selected by the evaluator.

- ***Entropy input:*** *the length of the entropy input value must equal the seed length.*
- ***Nonce:*** *If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length.*
- ***Personalization string:*** *The length of the personalization string must be ≤ seed length. If the implementation only supports one personalization string length, then the same length can be used for both values. If more than one string length is support, the evaluator shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.*
- ***Additional input:*** *the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.*

FCS_SLT_EXT.1 Cryptographic Salt Generation

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall ensure the TSS describes how salts are generated using the RBG.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall confirm by testing that the salts obtained in the cryptographic operations that use the salts are of the length specified in FCS_SLT_EXT.1, are obtained from the RBG, and are fresh on each invocation.

Note: in general these tests may be carried out as part of the tests of the relevant cryptographic operations.

FCS_STG_EXT.1 Protected Storage

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall review the TSS to determine that the TOE implements the required protected storage. The evaluator shall ensure that the TSS contains a description of the protected storage mechanism that justifies the selection of mutable hardware-based or software-based.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the process for generating keys, importing keys, or both, based on what is claimed by the ST. The evaluator shall also examine the operational guidance to ensure that it describes the process for destroying keys that have been imported or generated.

KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall test the functionality of each security function as described below. If the TOE supports both import and generation of keys, the evaluator shall repeat the testing as needed to demonstrate that the keys resulting from both operations are treated in the same manner. The devices used with the tooling may need to be non-production devices in order to enable the execution and gathering of evidence.

- ***Test 1:*** *The evaluator shall import or generate keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that generates a key/secret of each supported type and calls the import functions. The evaluator shall verify that no errors occur during import.*
- ***Test 2:*** *The evaluator shall write, or the developer shall provide access to, an application that uses a generated or imported key/secret:*
  - *For RSA, the secret shall be used to sign data.*
  - *For ECDSA, the secret shall be used to sign data.*

  The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported keys/secrets or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to use the key/secret imported or generated by the user or by a different application:
  - *The evaluator shall deny the approvals to verify that the application is not able to use the key/secret as described.*
  - *The evaluator shall repeat the test, allowing the approvals to verify that the application is able to use the key/secret as described.*

  If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.
- ***Test 3:*** *The evaluator shall destroy keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that destroys an imported or generated key/secret. The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to destroy the key/secret imported by the administrator or by a different application:*
  - *The evaluator shall deny the approvals and verify that the application is still able to use the key/secret as described.*
  - *The evaluator shall repeat the test, allowing the approvals and verifying that the application is no longer able to use the key/secret as described.*

If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

FCS_STG_EXT.2 Key Storage Encryption

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall review the TSS to determine that the TSS describes the protection of symmetric keys, KEKs, long-term trusted channel key material, and software-based key storage as claimed in FCS_STG_EXT.2.1.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component
Tests
There are no test evaluation activities for this component.

FCS_STG_EXT.3 Key Integrity Protection

This is an optional component. However, applied modules or packages might redefine it as mandatory.

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:
TSS
The evaluator shall examine the TSS and ensure that it contains a description of how the TOE protects the integrity of its keys.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
There are no test evaluation activities for this component.

5.1.3 User Data Protection (FDP)

FDP_TEE_EXT.1 Trusted execution environment for tenant software

> ***This is an optional component. However, applied modules or packages might redefine it as mandatory.***

FDP_TEE_EXT.1.1

The TSF shall implement a trusted execution environment that conforms to the following standard: [*Advanced Trusted Environment: OMTP TR1 v1.1*] and make this TEE available to tenant software.

Evaluation Activities

> *FDP_TEE_EXT.1:*

5.1.4 Class: Protection of the TSF (FPT)

FPT_ROT_EXT.1 Platform Integrity Root

FPT_ROT_EXT.1.1

The integrity of platform firmware shall be rooted in [**selection**:

- *code or data written to immutable memory or storage,*
- *credentials held in immutable storage on-platform or protected storage off-platform,*
- *a separate management controller that is itself rooted in a mechanism that meets this requirement,*
- *integrity measurements held securely in an on-platform dedicated security component,*
- *integrity measurements held securely by an off-platform entity*

].

Application Note: Roots of Trust are components that constitute a set of unconditionally trusted functions. The above are acceptable roots of trust for platform firmware integrity. The ST author must select the root of trust used to ensure the integrity of the first platform firmware that executes. The integrity of subsequently executed platform firmware must be traceable back to this root or to some other root as specified in FPT_ROT_EXT.2. This SFR should be iterated for additional TOE roots (for example, a management controller or firmware executed from an add-in card).

Evaluation Activities

> *FPT_ROT_EXT.1:*
> ***TSS***
> *The evaluator shall verify that the TSS describes the Root of Trust on which initial integrity of platform firmware is anchored, consistent with the selection above. The description shall include means by which the Root of Trust is protected from modification.*

FPT_ROT_EXT.2 Platform Integrity Extension

FPT_ROT_EXT.2.1

The integrity of all mutable platform firmware outside of the platform integrity root specified in FPT_ROT_EXT.1 shall be verified prior to execution or use through: [**selection**:

- *computation and verification of a hash by trusted code/data,*
- *verification of a digital signature by trusted code/data,*
- *measurement and verification by trusted code/data,*
- *measurement and verification by an on-platform dedicated security component,*
- *measurement and verification by an off-platform entity*

].

Application Note: This requirement specifies the means for extending the initial integrity of platform firmware established by FPT_ROT_EXT.1.1 to subsequently executed platform firmware and data that is located in mutable storage. (Integrity of code and data written to immutable storage is assured).

Otherwise, integrity must be extended through cryptographic means: either through hashes or digital signatures computed and verified by firmware that is trusted because it has previously had its integrity verified or is itself a Root of Trust. Verification can be performed by TOE components such as management controllers or non-TOE trusted entities.

FPT_ROT_EXT.2.2

The TOE shall take the following actions if an integrity check specified in FPT_ROT_EXT.2.1 fails:

1. Halt,
2. Notify an administrator/user by [**selection**: *generating an audit event*, *[**assignment**: other notification method(s)]*], and
3. [**selection**:
    - *Stop all execution and shut down,*

- *Initiate a Recovery process as specified in FPT_RVR_EXT.1*

]

[**selection**:

- *automatically,*
- *in accordance with administrator-configurable policy,*
- *by express determination of an administrator/user*

].

Application Note: Notification of an administrator can take many forms. For server-class platforms, such notification could take the form of administrator alerts or audit events. For platforms without management controllers, notification could be achieved, for example, by blinking lights, beep codes, screen indications, or local logging. If "*generating an audit event*" is selected then FAU_GEN.1 must be included in the ST.

Evaluation Activities

FPT_ROT_EXT.2:
TSS
The evaluator shall verify that the TSS describes the means by which initial integrity of platform firmware is extended to other platform components, and that the means are consistent with the selection(s) made in FPT_ROT_EXT.2. The TSF shall also describe how the TOE responds to failure of verification consistent with the selections in FPT_ROT_EXT.2.2.
Guidance
The evaluator shall examine the operational guidance to ensure that it describes the actions taken and notification methods used in case of failure to establish the integrity of the platform firmware root. If the actions are configurable, the guidance shall explain how they are configured.
Tests
The evaluator shall modify the platform firmware in a way that should cause a failure of the integrity check. The test passes if the mechanism specified in FPT_ROT_EXT.2.2 is triggered on the first subsequent boot of the platform.

Depending on the protections implemented, the evaluator may need a specially crafted update module from the vendor to perform this test. But note that this is not necessarily the same as a test of the update mechanism. The update mechanism can be tested either at boot time or at the time of the update. This verification check must be done during boot.

If modification of platform firmware in situ or using the update mechanism is deemed to be not feasible within the time and cost constraints of the evaluation, then the evaluators shall make such an argument in the AAR, and with concurrence of the CC scheme, this test can be replaced by evidence of vendor testing.

FPT_PHP.1 Passive detection of physical attack

This is an optional component. However, applied modules or packages might redefine it as mandatory.

FPT_PHP.1.1
The TSF shall provide unambiguous detection of physical tampering that might compromise the TSF.

FPT_PHP.1.2
The TSF shall provide the capability to determine whether physical tampering with the TSF's devices or TSF's elements has occurred.

Evaluation Activities

FPT_PHP.1:

FPT_PHP.2 Notification of physical attack

This is an optional component. However, applied modules or packages might redefine it as mandatory.

FPT_PHP.2.1
The TSF shall provide unambiguous detection of physical tampering that might compromise the TSF.

FPT_PHP.2.2
The TSF shall provide the capability to determine whether physical tampering with the TSF's devices or TSF's elements has occurred.

FPT_PHP.2.3
For [**assignment**: *list of TSF devices/elements for which active detection is required*], the TSF shall monitor the devices and elements and notify [**assignment**: *a designated user or role*] when physical tampering with the TSF's devices or TSF's elements has occurred.

Evaluation Activities

FPT_PHP.2:

FPT_PHP.3 Resistance to physical attack

This is an optional component. However, applied modules or packages might redefine it as mandatory.

FPT_PHP.3.1

The TSF shall resist [**assignment**: *physical tampering scenarios*] to the [**assignment**: *list of TSF devices/elements*] by responding automatically such that the SFRs are always enforced.

Evaluation Activities

FPT_PHP.3:
TSS
The evaluator shall examine the TSS to ensure it describes the methods used by the TOE to detect physical tampering and how the TOE will respond when physical tampering has been detected.

Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
Need to design a test to detect and resist tamper.

FPT_PHP_EXT.1 Hardware component integrity

This is an objective component.

FPT_PHP_EXT.1.1

Outside of the integrity root specified in FPT_ROT_EXT.1, the integrity of [**assignment**: *critical platform hardware components*]shall be verified prior to execution or use through: [**assignment**: *method for ensuring integrity of platform hardware components*].

Application Note: The purpose of this objective requirement is to encourage platform and component vendors to adopt mechanisms similar to those defined in upcoming NIST SP 1800-34 for ensuring the integrity of the hardware supply chain. The scope of SP 1800-34 is to cover "manufacturing and OEM processes that protect against counterfeits, tampering, and insertion of unexpected software and hardware, and the corresponding customer processes that verify that client and server computing devices and components have not been tampered with or otherwise modified. Manufacturing processes that cannot be verified by the customer are explicitly out of scope."

As a basic step, critical platform components should include immutable hardware IDs that can be listed a hardware component manifest that is provided to the purchaser and signed by the manufacturer. It should then be possible for the TOE to verify the signature on the manifest and check that each hardware ID in the manifest matches the IDs in the actual hardware.

For purposes of this requirement, hardware identities can be verified once on first boot, on every boot, when new hardware is detected, or during normal operation of the platform - as long as the hardware integrity is verified before the component or device is used.

The ST author lists the hardware components for which the integrity is checked, and the methods used for conducting the checks. "Critical components" generally would include chassis, motherboards, CPUs, network cards, memory chips, hard drives, controllers, graphics processors, and service controllers.

FPT_PHP_EXT.1.2

The TOE shall take the following actions if an integrity check specified in FPT_PHP_EXT.1.1 fails:

1. Halt,
2. Notify an administrator/user by [**selection**: *generating an audit event*, *[**assignment**: other notification method(s)]*], and
3. [**selection**:
    - *Stop all execution and shut down*,
    - *Continue execution without the integrity-compromised component*,
    - *Continue execution*

    ]

[**selection**:

- *in accordance with administrator-configurable policy*,
- *by express determination of an administrator/user*

].

Application Note: Notification of an administrator can take many forms. For server-class platforms, such notification could take the form of administrator

alerts or audit events. For platforms without management controllers, notification could be achieved, for example, by blinking lights, beep codes, screen indications, or local logging. If "*generating an audit event*" is selected then FAU_GEN.1 must be included in the ST.

Evaluation Activities

FPT_PHP_EXT.1:

FPT_PPF_EXT.1 Protection of Platform Firmware and Critical Data

FPT_PPF_EXT.1.1

The TSF shall allow modification of platform firmware only through the update mechanisms described in FPT_TUD_EXT.1.

Application Note: Platform firmware must be modifiable only through one of the secure update mechanisms specified in FPT_TUD_EXT.1. If the update mechanism itself is implemented in platform firmware, then naturally, it must itself also be modifiable only through the secure update mechanism. Configuration data used by platform firmware that is stored in nonvolatile memory is not included in these protections. Software portions of TSF and data critical for ensuring the integrity of the TSF are included in these protections. Specifically, this includes the key store and the signature verification algorithm used by the update mechanisms.

Evaluation Activities

FPT_PPF_EXT.1:
TSS
The evaluator shall examine the TSS to ensure that it explains how the various areas of platform firmware and critical data are protected from modification outside of the platform firmware update mechanism described in FPT_TUD_EXT.1. If the TOE implements an authenticated update mechanism as specified in FPT_TUD_EXT.2, then the evauator shall ensure that the TSS describes specifically how the signature verification code and key store is protected from update outside of the secure platform firmware update mechanism.
Guidance
The evaluator shall check the operational guidance to ensure that there are instructions for how to securely modify the platform firmware and critical data using a mechanism specified in FPT_TUD_EXT.1.
Tests

- ***Test 1:*** *The evaluator shall attempt to overwrite or modify the platform firmware without invoking one of the update mechanisms specified in FPT_TUD_EXT.1 (e.g., using a modified Linux boot loader such as GRUB that attempts to write to the memory where platform firmware is stored). The test succeeds if the attempts to overwrite platform firmware fail. The evaluator shall attempt at least two such tests--one that attempts to overwrite the first platform firmware that executes after boot, and one that targets the secure update mechanism (if implemented), and one that targets firmware that has been integrity-checked since the last boot.*

FPT_RVR_EXT.1 Platform Firmware Recovery

This is a selection-based component. Its inclusion depends upon selection from FPT_ROT_EXT.2.2.

FPT_RVR_EXT.1.1

The TSF shall implement a mechanism for recovering from boot firmware failure consisting of [**selection**:

- *the secure local update mechanism described in FPT_TUD_EXT.4*,
- *installation of a known-good or recovery firmware image*,
- *reversion to the prior firmware image*,
- *installation of a recovery image that puts the TOE into a maintenance mode*

]

Application Note: This SFR must be included in the ST if:

- "Initiate a Recovery process as specified in FPT_RVR_EXT.1" is selected in FPT_ROT_EXT.2.2,
- "Initiate a Recovery process as specified in FPT_RVR_EXT.1" is selected in FPT_TUD_EXT.2.5,
- The TOE implements a recovery mechanism for firmware corruption not necessarily related to integrity or update failure.

As indicated above, in addition to integrity or update failure, the TOE may use a recovery mechanism to deal with non-security-related failures, such as a power outage during update or a power surge during normal operation.

The recovery process may be initiated automatically on failure, as the result of physically present User action, or as the result of pre-configured policy. The action taken may depend on the nature of the failure as specified in FPT_ROT_EXT.2.2 and FPT_TUD_EXT.2.5.

Evaluation Activities

FPT_RVR_EXT.1:
TSS
The evaluator shall examine the TSS section to confirm that it describes how the platform firmware recovery mechanism works and the conditions under which it is invoked.
Guidance
The evaluator shall examine the guidance to ensure that is describes how to configure the conditions under which the recovery mechanism is initiated (if configurable).
Tests
The evaluators shall perform the following tests:

- ***Test 1:*** *To test this requirement, the evaluator shall trigger the recovery process either by forcing an update error or a boot integrity failure and observing that the recovery process has been initiated.*
- ***Test 2:*** *The evaluator will engage with the recovery process as necessary, and after recovery will determine the version of the current firmware image. The test is passed if the resultant image is as expected in accordance with policy and the selections in FPT_RVR_EXT.1.1. If the recovery process uses the secure local update process as specified in FPT_TUD_EXT.4, then this test is satisfied by testing of that requirement.*

FPT_TUD_EXT.1 TOE Firmware Update

FPT_TUD_EXT.1.1

The TSF shall [**selection**:

- *make no provision for platform firmware update,*
- *implement an authenticated platform firmware update mechanism as described in FPT_TUD_EXT.2,*
- *implement an unauthenticated platform firmware update mechanism as described in FPT_TUD_EXT.3,*
- *implement a secure local platform firmware update mechanism described in FPT_TUD_EXT.4*

].

Application Note: The purpose of the platform firmware update mechanism is to ensure the authenticity and integrity of platform firmware updates. If platform firmware is immutable (not updateable by any non-destructive means) then the ST author must select "make no provision for platform firmware update."

If platform firmware is modifiable only through a local update requiring physical presence at the platform, then the ST author must select "implement a secure local update process..." and include FPT_TUD_EXT.4 in the ST.

If the platform implements an update mechanism that does not require physical presence at the platform, and that authenticates firmware updates prior to installing them, then the ST author selects "implement an authenticated platform update mechanism..." and include FPT_TUD_EXT.2 in the ST.

If the platform implements an update mechanism that does not require physical presence at the platform, and that does not authenticate firmware updates prior to installing them, then the ST author selects "implement an unauthenticated platform update mechanism..." and include FPT_TUD_EXT.3 in the ST.

Evaluation Activities

FPT_TUD_EXT.1:
TSS
If the ST author selects "make no provision for platform firmware update," then the evaluator shall examine the TSS to ensure that it explains all ways of modifying platform firmware in the absence of any provided mechanism. For example, breaking open the case and prying a chip off the motherboard and then reprogramming the chip. The purpose of this activity is to ensure that the TOE does not implement a local update mechanism that does not meet the requirements of FPT_TUD_EXT.4.

This requirement is met if the platform implements no means for updating platform firmware and the TSS describes a method for updating or replacing platform firmware that involves potentially destroying or damaging the TOE or some of its components.

If the ST author selects "implement an authenticated platform firmware update mechanism..." then this requirement is satisfied if FPT_TUD_EXT.2 is satisfied.

If the ST author selects "implement an unauthenticated platform firmware update mechanism..." then this requirement is satisfied if FPT_TUD_EXT.3 is satisfied.

If the ST author selects "implement a secure local platform update mechanism..." then this requirement is satisfied if FPT_TUD_EXT.4 is satisfied.

FPT_TUD_EXT.2 Platform Firmware Authenticated Update Mechanism

This is a selection-based component. Its inclusion depends upon selection from FPT_TUD_EXT.1.1.

FPT_TUD_EXT.2.1

The TSF shall authenticate the source of all platform firmware updates using a digital signature algorithm specified in FCS_COP.1/SigVer and using a key store that contains [**selection**: *the public key, hash value of the public key*].

Application Note: The ST must include FCS_COP.1/Hash if "hash value of the

public key" is selected.

FPT_TUD_EXT.2.2

The TSF shall allow installation of updates only if the digital signature has been successfully verified as specified in FCS_COP.1/SigVer and [**selection**: *the version number of the platform firmware update is more recent than the version number of the current installed platform firmware, no other conditions*].

Application Note: The ST author should make the selection above if the TSF supports rollback prevention. That is, the TSF does not allow "update" to an older version of the platform firmware. In general, rollback should be permitted only through a secure local update mechanism at the express direction of an Administrator/User.

FPT_TUD_EXT.2.3

The TSF shall include a platform firmware version identifier that is accessible by the update mechanism and includes information that enables the update mechanism to determine the relative order of updates.

FPT_TUD_EXT.2.4

The TSF shall provide an observable indication of the success or failure of the update operation.

Application Note: For success, this indication should include the version number of the newly installed firmware. Notification of failure could include generation of an audit event by a management subsystem, a beep code, an updated version number on a splash screen, or simple failure to continue functioning.

FPT_TUD_EXT.2.5

The TOE shall take the following actions if a platform firmware integrity, authenticity, or rollback-prevention check fails, or a platform firmware update fails for any other reason:

- Do not install the update,
- Notify an administrator/user by [**selection**: *generating an audit event, [**assignment**: notification method]*]

, and [**selection**:

- *Continue execution,*
- *Halt,*
- *Stop all execution and shut down,*
- *Initiate recovery as specified in FPT_RVR_EXT.1*

] [**selection**:

- *automatically,*
- *in accordance with administrator-configurable policy,*
- *by express determination of an administrator/user*

].

Application Note: The platform firmware authenticated update mechanism employs digital signatures to ensure the authenticity of the firmware update image. The TSF includes a signature verification algorithm and a key store containing the public key needed to verify the signature on the firmware update image.

A hash of the public key may be stored if a copy of the public key is provided with firmware update images. In this case, the update mechanism shall hash the public key provided with the update image, and ensure that it matches a hash which appears in the key store before using the provided public key to verify the signature of the update image. If the hash of the public key is selected, the ST author may iterate the FCS_COP.1/Hash requirement to specify the hashing functions used.

An indication of success or failure can be generation of an audit event by a management subsystem, a beep code, an updated version number on a splash screen, or simple failure to continue functioning.

If the update mechanism generates audit events, the ST author shall make the appropriate selections from the audit events table (Table t-audit-sel-based).

Evaluation Activities

FPT_TUD_EXT.2:

TSS
The evaluator shall ensure that the TSS includes a comprehensive description of how the authentication of platform firmware updates is implemented by the TSF. The TSS should cover the initialization process and the activities that are performed to ensure that the digital signature of the update image is verified before modification of the firmware.

The evaluator shall examine the TSF to ensure that it describes the platform firmware version identifier and explains its meaning and encoding.

The evaluator shall also ensure that the TSS describes the actions taken by the TSF is an update image fails authentication.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the process for updating the platform firmware.

The evaluator shall examine the operational guidance to ensure that it documents the observable indications of update success or failure, and that it describes how to access the platform

firmware version indicators.

Tests

- ***Test 1:*** *The evaluator determines the current version of the platform firmware, and obtains or produces a valid, authentic, and permissible update image of platform firmware. The evaluator initiates an update using this image through the process described in the operational guidance. After the process is complete, the evaluator checks the current firmware version to ensure that the new firmware version matches that of the update.*
- ***Test 2:*** *The evaluator performs the same test, this time using a valid update image that is signed with an incorrect key. The update must fail.*
- ***Test 3:*** *The evaluator performs the same test, this time using an update image that is corrupted but is signed with the correct key. The update must fail.*
- ***Test 4:*** *The evaluator performs the same test, this time using a valid update image that is not signed. The update must fail.*
- ***Test 5:*** *If the TSF implements rollback protections, the evaluator performs the same test, this time using a valid, signed update image that is has an earlier version number than the currently installed firmware. The update must fail.*

FPT_TUD_EXT.3 Platform Firmware Delayed-Authentication Update Mechanism

This is a selection-based component. Its inclusion depends upon selection from FPT_TUD_EXT.1.1.

FPT_TUD_EXT.3.1

The TSF shall allow execution or use of platform firmware updates only if new platform firmware is integrity- and authenticity-checked using the mechanism described in FPT_ROT_EXT.2 prior to its execution or use, and [**selection**: *the version number of the platform firmware update is more recent than the version number of the previously installed platform firmware, no other conditions*].

Application Note: This update mechanism does not require an integrity or authenticity check prior to installation, but the newly installed platform firmware must have its integrity and authenticity verified prior to being executed or used. This update mechanism takes advantage of the existing FPT_ROT_EXT.2 requirement to avoid having to verify the integrity and authenticity of an update package at install time.

The ST author should select "the version number of the platform firmware update is more recent than the version number of the previously installed platform firmware" if the TSF supports rollback prevention.

FPT_TUD_EXT.3.2

The TSF shall include an observable platform firmware version identifier that is accessible by the update mechanism and includes information that enables the update mechanism to determine the relative order of updates.

FPT_TUD_EXT.3.3

The TSF shall provide an observable indication of the success or failure of the update operation.

Application Note: For success, this should at least include an indication of the version number of the newly installed firmware. Notification of failure could include generation of an audit event by a management subsystem, a beep code, an updated version number on a splash screen, or simple failure to continue functioning.

FPT_TUD_EXT.3.4

The TOE shall take the following actions if a platform firmware update integrity, authentication, or rollback-prevention check fails, or a platform firmware update fails for any other reason:

- Notify Notify an administrator/user by [**selection**: *generating an audit event*, ***[assignment****: notification method]*]

and [**selection**:

- *Halt*,
- *Stop all execution and shut down*,
- *Initiate a recovery process as specified in FPT_RVR_EXT.1*

] [**selection**:

- *automatically*,
- *in accordance with administrator-configurable policy*,
- *by express determination of an administrator/user*

].

Application Note: The platform firmware unauthenticated update mechanism installs platform firmware updates without first checking their integrity or authenticity. Instead, this mechanism either invokes a special authentication/integrity check on the firmware *in situ* after install or relies on the firmware checks required by FPT_ROT_EXT.2 to ensure the integrity and authenticity of the update image. In either case, the integrity and authenticity of the update must be verified before the updated firmware is executed or used.

Likewise, if the TSF implement rollback prevention, this check must be made before the newly installed firmware is executed.

Evaluation Activities

FPT_TUD_EXT.3:

TSS
The evaluator shall ensure that the TSS includes a comprehensive description of how the authentication of platform firmware updates is implemented by the TSF. The TSS should cover the initialization process and the activities that are performed to ensure that the digital signature of the update image is verified before it is executed or used.

The evaluator shall examine the TSF to ensure that it describes the platform firmware version identifier and explains its meaning and encoding.

The evaluator shall also ensure that the TSS describes the actions taken by the TSF if an update image fails authentication, integrity, or rollback-prevention checks.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the process for updating the platform firmware.

The evaluator shall examine the operational guidance to ensure that it documents the observable indications of update success or failure, and that it describes how to access the platform firmware version indicators.

Tests

- ***Test 1:*** *The evaluator determines the current version of the platform firmware, and obtains or produces a valid, authentic, and permissible update image of platform firmware. The evaluator initiates an update using this image through the process described in the operational guidance. After the process is complete, the evaluator checks the current firmware version to ensure that the new firmware version matches that of the update.*
- ***Test 2:*** *The evaluator performs the same test, this time using a inauthentic update image. The update code must fail to execute.*
- ***Test 3:*** *The evaluator performs the same test, this time using an update image that is corrupted but is otherwise authentic. The update code must fail to execute.*
- ***Test 4:*** *If the TSF implements rollback protections, the evaluator performs the same test, this time using a valid, signed update image that is has an earlier version number than the currently installed firmware. The update code must fail to execute.*

FPT_TUD_EXT.4

This is a selection-based component. Its inclusion depends upon selection from FPT_TUD_EXT.1.1.

FPT_TUD_EXT.4.1

The TSF shall provide a secure local update mechanism that requires an assertion of physical access to the TOE before installation of an update.

FPT_TUD_EXT.4.2

The Administrator/User shall assert physical presence to the TSF through:
[**selection**:

- *login to the TOE from a physically connected console or terminal,*
- *physical connection of a jumper or cable,*
- *connection to a debug port,*
- *[**assignment**: description of other mechanism for asserting physical presence]*

]

Application Note: This requirement pertains to platform firmware update mechanisms that do not use the authentication-based update mechanism described in FPT_TUD_EXT.2 or the delayed-authentication described in FPT_TUD_EXT.3. The secure local update mechanism ensures the authenticity and integrity of the firmware update image by requiring an Administrator/User to be physically present at the TOE. An assertion of physical presence can take the form, for example, of requiring entry of a password at a boot screen, unlocking of a physical lock (e.g., a motherboard jumper), or inserting a USB cable before permitting platform firmware to be updated.

There is no requirement that the local update mechanism support rollback prevention.

The local update mechanism must be a designed mechanism. If update can be accomplished only through the physical removal and replacement of a part, then that is not a secure local update mechanism--that is no update mechanism--and "make no provision for platform firmware update" should be selected in FPT_TUD_EXT.1.1.

FPT_TUD_EXT.4.3

The TSF shall include a platform firmware version identifier that is accessible by the update mechanism or to the Administrator/User who asserts physical presence.

FPT_TUD_EXT.4.4

The TSF shall provide an observable indication of the success or failure of the update operation.

Application Note: For success, this indication should include the version number of the newly installed firmware. Notification of failure could be through a beep code, an indication on a splash screen, or simple failure to continue functioning.

Evaluation Activities

FPT_TUD_EXT.4:
TSS
The evaluator shall check the TSS section to confirm that it clearly and thoroughly describes how the secure local update functionality is implemented.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes instructions for using the local update mechanism, and how to validate that the update was succesful.

Tests
- ***Test 1:*** *The evaluator tests the secure local update by following the instructions provided in the operational guidance to update the platform firmware image. The update must succeed.*
- ***Test 2:*** *The evaluator next tries to update the platform firmware image without first asserting physical presence. The update must fail or be not possible.*

5.1.5 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 to be performed are specified both in Section 5 Security 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 ITSEF 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 Security 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 Security 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 .

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 . 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 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 - Implementation-Dependent Requirements

Implementation-Dependent 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 Widget Thing

If the TOE includes the widget thing, all of the following SFRs must be claimed:

- FQQ_QQQ.6

If this is implemented by the TOE, the following requirements must be included in the ST:

Appendix B - Acronyms

Appendix C - 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 D - Use Case Templates

D.1 Server-Class Platform

D.2 Server-Class Platform, Enhanced

D.3 Portable Clients (laptops, tablets)

D.4 Portable Clients (laptops, tablets), Enhanced

D.5 CSfC EUD

D.6 CSfC Hardened EUD

D.7 Tactical EUD

D.8 Desktop clients

D.9 Thin/Zero Clients

D.10 IoT Devices

Appendix E - 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 F - 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.