collaborative Protection Profile for Full Drive Encryption – Encryption Engine

Version: 3.0
2026-03-12
Full Disk Encryption international Technical Community

Revision History

Version	Date	Comment
0.1	2014-08-26	Initial release for iTC review
0.2	2014-09-05	Draft published for public review
0.13	2014-10-17	Incorporated comments received from the public review
1.0	2015-01-26	Incorporated comments received from the CCDB review
1.5	2015-09-02	Revised based on additional use cases developed by iTC
2.0	2016-09-09	Incorporated comments received from the public review, and also updated the Key Destruction section and AVA_VAN.
2.0 + Errata 20190201	2019-02-01	Updated to reflect CC Part 3 evaluation findings and FDE Interpretation Team [FIT] rulings
3.0	2026-03-12	Updated for CC:2022, reviewing and applying modifications to the cPP

Contents

1 Introduction

1.1 PP Overview

The purpose of the set of Collaborative Protection Profiles (cPPs) for Full Drive Encryption (FDE): Authorization Acquisition (AA) and Encryption Engine (EE) is to provide requirements for Data-at-Rest protection against unauthorized access or disclosure of stored data on a lost device. These cPPs allow FDE solutions based in software and/or hardware to meet the requirements for Data-at-Rest protection. The form factor for a storage device may vary, but could include: hard disk drives/solid state drives in servers, workstations, laptops, mobile devices, tablets, and external media. A hardware solution could be a Self-Encrypting Drive or other hardware-based solutions; the interface (USB, SATA, etc.) used to connect the storage device to the host machine is outside the scope of this cPP.

Full Drive Encryption encrypts all data (with certain exceptions) on the storage device and permits access to the data only after successful authorization to the FDE solution. The exceptions include the necessity to leave a portion of the storage device (the size may vary based on implementation) unencrypted for such things as the Master Boot Record (MBR) or other AA/EE pre-authentication software. These FDE cPPs interpret the term “full drive encryption” to allow FDE solutions to leave a portion of the storage device unencrypted so long as it does not contain plaintext user or plaintext authorization data.

Since the FDE cPPs support a variety of solutions, two cPPs describe the requirements for the FDE components shown in Figure 1.

Figure 1: FDE Components

The FDE cPP - Authorization Acquisition describes the requirements for the Authorization Acquisition piece and details the necessary security requirements and assurance activities necessary to interact with a user and result in the availability of a Border Encryption Value (BEV).

The FDE cPP - Encryption Engine describes the requirements for the Encryption Engine piece and details the necessary security requirements and assurance activities for the actual encryption and decryption of the data by the DEK. Each cPP will also have a set of core requirements for management functions, proper handling of cryptographic keys, updates performed in a trusted manner, audit and self-tests.

The Target of Evaluation (TOE) description defines the scope and functionality of the Encryption Engine, and the Security Problem Definition describes the assumptions made about the operating environment and the threats to the EE that the cPP requirements address.

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 (Base-PP)	Protection Profile used as a basis to build a PP-Configuration.
Collaborative Protection Profile (cPP)	A Protection Profile developed by international technical communities and approved by multiple schemes.
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.
Direct Rationale	A type of Protection Profile, PP-Module, or Security Target in which the security problem definition (SPD) elements are mapped directly to the SFRs and possibly to the security objectives for the operational environment. There are no security objectives for the TOE.
Extended Package (EP)	A deprecated document form for collecting SFRs that implement a particular protocol, technology, or functionality. See Functional Packages.
Functional Package (FP)	A document that collects SFRs for a particular protocol, technology, or functionality.
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-PPs.
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.
Target of Evaluation (TOE)	The product under evaluation.
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.

1.2.2 Technical Terms

Authorization Factor	A value that a user knows, has, or is (e.g. password, token, etc.) submitted to the TOE to establish that the user is in the community authorized to use the drive. This value is used in the derivation or decryption of the BEV and eventual decryption of the DEK. Note that these values may or may not be used to establish the particular identity of the user.
Authorized User	A user who has a valid Authorization Factor or legitimate physical possession of the drive.
Border Encryption Value (BEV)	A value passed from the FDE Authorization Acquisition (AA) to the FDE Encryption Engine (EE) intended to link the key chains of the two components.
Data Encryption Key (DEK)	A key used to encrypt data-at-rest.
Full Drive Encryption (FDE)	Refers to partitions of logical blocks of user accessible data as managed by the host system. FDE products encrypt all data (with certain exceptions) on the partition of the storage device and permits access to the data only after successful authorization. FDE solutions may leave a portion of the storage device unencrypted, for such things as the Master Boot Record (MBR) or other AA/EE pre-authentication software, so long as it contains no protected data.
Intermediate Key	A key used in a point between the initial user authorization and the DEK.
Key Chaining	The method of using multiple layers of encryption keys to protect data. A top layer key encrypts a lower layer key which encrypts the data; this method can have any number of layers.
Key Encryption Key (KEK)	A key used to encrypt other keys, such as DEKs or storage that contains keys.
Key Material	Key material is commonly known as critical security parameter (CSP) data, and also includes authorization data, nonces, and metadata.
Key Release Key (KRK)	A key used to release another key from storage, it is not used for the direct derivation or decryption of another key.
Key Sanitization	A method of sanitizing encrypted data by securely overwriting the key that was encrypting the data.
Non-Volatile Memory	A type of computer memory that will retain information without power.
Operating System (OS)	Software which runs at the highest privilege level and can directly control hardware resources.
Powered-Off State	The device has been shut down.
Protected Data	This refers to all encrypted data on the storage device. Protected data may exclude a Master Boot Record or Pre-authentication area of the drive – areas that are sometimes necessarily unencrypted.

Root of Trust for Update (RTU)	A type of RoT that verifies the integrity and authenticity of an update payload before initiating the update process.
Submask	A submask is a bit string that can be generated and stored in a number of ways. The individual SFRs provide context where these values may be used and what functions they perform, such as the BEV, intermediate key, or derived authentication value.

1.3 Implementation

Full Drive Encryption solutions vary with implementation and vendor combinations.

Vendors must evaluate products that provide both components of the Full Disk Encryption Solution (AA and EE) against both cPPs, although that could be done in a single evaluation with one ST. A vendor that provides a single component of an FDE solution would only evaluate against the applicable cPP. The FDE cPP is divided into two documents to allow labs to independently evaluate solutions tailored to one cPP or the other. When a customer acquires an FDE solution, they will either obtain a single vendor product that meets the AA + EE cPPs or two products, one of which meets the AA and the other of which meets the EE cPPs.

Table 1 illustrates a few examples for certification

Table 1: Examples of cPP Implementations

Implementation	cPP	Description
Host	AA	Host software provides the interface to a self-encrypting drive
Self-Encrypting Drive (SED)	EE	A self-encrypting drive used in combination with separate host software
Software FDE	AA + EE	A software full drive encryption solution
Hybrid	AA + EE	A single vendor’s combination of hardware (e.g., hardware encryption engine, cryptographic co-processor) and software / firmware

1.4 TOE Overview

The Target of Evaluation (TOE) for this cPP is either the Encryption Engine or a combined evaluation of the set of cPPs for FDE (Authorization Acquisition or Encryption Engine).

The following sections provide an overview of the functionality of the FDE EE as well as the security capabilities.

1.4.1 Encryption Engine Introduction

The Encryption Engine (EE) objectives focus on data encryption, policy enforcement, and key management. The EE is responsible for the generation, update, archival, recovery, protection, and destruction of the DEK and other intermediate keys under its control. The EE receives a Border Encryption Value (BEV) from the AA. The EE uses that BEV for the decryption of the DEK although other intermediate keys may exist in between those two points. Key Encryption Keys (KEKs) wrap other keys, notably the DEK or other intermediary keys which chain to the DEK. Key Releasing Keys (KRKs) authorize the EE to release either the DEK or other intermediary keys which chain to the DEK. These keys only differ in the functional use.

The EE determines whether to allow or deny a requested action based on the KEK or KRK provided by the AA. Possible requested actions include but are not limited to changing of encryption keys, decryption of data, and key sanitization of encryption keys (including the DEK). The EE may offer additional policy enforcement to prevent access to ciphertext or the unencrypted portion of the storage device. Additionally the EE may provide encryption support for multiple users on an individual basis.

Figure 2 illustrates the components within EE and its relationship with AA.

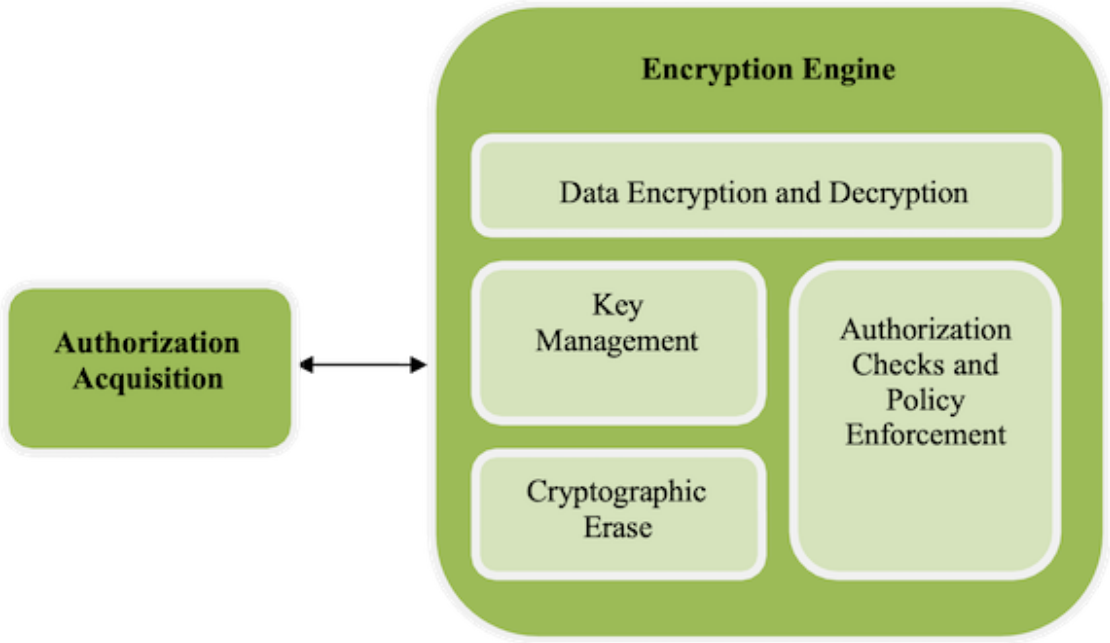


Figure 2: Encryption Engine Details

1.4.2 Encryption Engine Security Capabilities

The Encryption Engine is ultimately responsible for ensuring that the data is encrypted using a prescribed set of algorithms. The EE manages the decryption of the data on the storage device through decryption of the DEK based on the validity of the BEV provided by the AA. It also manages administrative functions, such as changing the DEK, managing the BEVs required for decrypting or releasing the DEK, managing the intermediate wrapping keys under its control, and performing a key sanitization.

The EE may provide key archiving and recovery functionality. The EE may manage the archiving and recovery itself, or interface with the AA to perform this function. It may also offer configurable features, which restricts the movement of keying material and disables recovery functionality.

The foremost security objective of encrypting storage devices is to force an adversary to perform an exhaustive search against a prohibitively large key space in order to recover the DEK or other intermediate keys. The EE uses approved cryptography to generate, handle, and protect keys to force an adversary who obtains an unpowered lost or stolen platform without the authorization factors or intermediate keys to exhaust the encryption key space of intermediate keys or DEK to obtain the data. The EE randomly generates DEKs and – in some cases - intermediate keys. The EE uses DEKs in a symmetric encryption algorithm in an appropriate mode along with appropriate initialization vectors for that mode to encrypt storage units (e.g. sectors or blocks) on the storage device. The EE either encrypts the DEK with a KEK or an intermediate key.

1.4.3 Interface/Boundary

The interface and boundary between the AA and the EE will vary based on the implementation. If one vendor provides the entire FDE solution, then it may choose to not implement an interface between the AA and EE components. If a vendor provides a solution for one of the components, then the assumptions below state that the channel between the

two components is sufficiently secure. Although standards and specifications exist for the interface between AA and EE components, the cPP does not require vendors to follow the standards in this version.

1.5 Compliant Targets of Evaluation

1.5.1 TOE Boundary

The environment in which the EE functions may differ depending on the boot stage of the platform in which it operates; see Figure 3. Aspects of initialization, and perhaps authorization may be performed in the Pre-Boot environment, while provisioning, encryption, decryption and management functionality are likely performed in the Operating System environment. Some of these aspects may occur in both environments. The Operating System environment may make a full range of services available to the Encryption Engine, including hardware drivers, cryptographic libraries, and perhaps other services external to the TOE. The Pre-Boot environment is much more constrained with limited capabilities. This environment turns on the minimum number of peripherals and loads only those drivers necessary to bring the platform from a cold start to executing a fully functional operating system with running applications. The EE TOE may include or leverage features and functions within the operational environment.

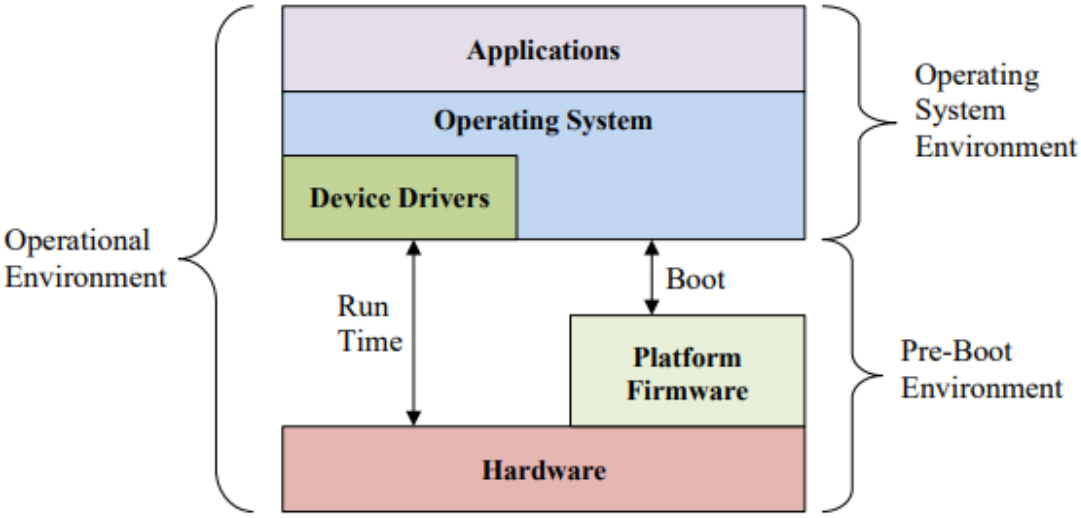


Figure 3: Operational Environment

1.6 Use Cases

The use case for a product conforming to the FDE cPPs is to protect data-at-rest on a device that is lost or stolen while powered off without any prior access by an adversary. The use case where an adversary obtains a device that is in a powered state and is able to make modifications to the environment or the TOE itself (e.g., evil maid attacks) is not addressed by these cPPs (i.e., FDE-AA and FDE- EE).

1.7 Product Features Mapped to Implementation-dependent Requirements

The features enumerated below, if implemented by the TOE, require that additional SFRs be claimed in the ST.

1.7.1 Symmetric Key Generation Support

A conformant TOE is required to implement proper generation of symmetric keys. This is selected based in FCS_CKM.1/DEK when the TSF generates a DEK. Or symmetric key generation may be used in key wrapping per FCS_COP.1/KeyWrap or key encryption per FCS_COP.1/KeyEnc.

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

- **FCS_CKM.1/SKG**

2 Conformance Claims

Conformance Statement

An ST must claim exact conformance to this PP.

The evaluation methods used for evaluating the TOE are a combination of the workunits defined in [CEM] as well as the Evaluation Activities for ensuring that individual SFRs and SARs have a sufficient level of supporting evidence in the Security Target and guidance documentation and have been sufficiently tested by the laboratory as part of completing ATE_IND.1. Any functional packages this PP claims similarly contain their own Evaluation Activities that are used in this same manner.

CC Conformance Claims

This PP is conformant to Part 2 (extended) and Part 3 (conformant) of Common Criteria CC:2022, Revision 1.

PP Claim

This PP does not claim conformance to any Protection Profile.

There are no PPs or PP-Modules that are allowed in a PP-Configuration with this PP.

Package Claim

This PP is not conformant to any Functional or Assurance Packages.

3 Security Problem Definition

3.1 Threats

This section provides a narrative that describes how the requirements mitigate the mapped threats. A requirement may mitigate aspects of multiple threats. A requirement may only mitigate a threat in a limited way. Some requirements are optional, either because the TSF fully mitigates the threat without the additional requirements being claimed or because the TSF relies on its Operational Environment to provide the functionality that is described by the optional requirements.

A threat consists of a threat agent, an asset and an adverse action of that threat agent on that asset. The threat agents are the entities that put the assets at risk if an adversary obtains a lost or stolen storage device. Threats drive the functional requirements for the Target of Evaluation (TOE). For instance, one threat below is T.UNAUTHORIZED_DATA_ACCESS. The threat agent is the possessor (unauthorized user) of a lost or stolen storage device. The asset is the data on the storage device, while the adverse action is to attempt to obtain those data from the storage device. This threat drives the functional requirements for the storage device encryption (TOE) to authorize who can use the TOE to access the hard disk and encrypt/decrypt the data. Since possession of the KEK, DEK, intermediate keys, authorization factors, submasks, and random numbers or any other values that contribute to the creation of keys or authorization factors could allow an unauthorized user to defeat the encryption, this SPD considers key material equivalent to the data in importance and they appear among the other assets addressed below.

It is important to reemphasize at this point that this collaborative Protection Profile does not expect the product (TOE) to defend against the possessor of the lost or stolen storage device who can introduce malicious code or exploitable hardware components into the Target of Evaluation (TOE) or the Operational Environment. It assumes that the user physically protects the TOE and that the Operational Environment provides sufficient protection against logical attacks. One specific area where a conformant TOE offers some protection is in providing updates to the TOE; other than this area, though, this cPP mandates no other countermeasures. Similarly, these requirements do not address the “lost and found” hard disk problem, where an adversary may have taken the hard disk, compromised the unencrypted portions of the boot device (e.g., MBR, boot partition), and then made it available to be recovered by the original user so that they would execute the compromised code.

T.AUTHORIZATION_GUESSING

Threat agents may exercise host software to repeatedly guess authorization factors, such as passwords and PINs. Successful guessing of the authorization factors may cause the TOE to release DEKs or otherwise put it in a state in which it discloses protected data to unauthorized users.

T.CHOSEN_PLAINTEXT

Threat agents may trick authorized users into storing chosen plaintext on the encrypted storage device in the form of an image, document, or some other file. A poor choice of encryption algorithms, encryption modes, and initialization vectors along with the chosen plaintext could allow attackers to recover the effective DEK, thus providing unauthorized access to the previously unknown plaintext on the storage device.

T.KEYING_MATERIAL_COMPROMISE

Possession of any of the keys, authorization factors, submasks, and random numbers or any other values that contribute to the creation of keys or authorization factors could allow an unauthorized user to defeat the encryption. The cPP considers possession of key material of equal importance to the data itself. Threat agents may look for keying material in unencrypted sectors of the storage device and on other peripherals in the operational environment (OE), (e.g., BIOS configuration, SPI flash, or TPMs).

T.KEYSPACE_EXHAUST

Threat agents may perform a cryptographic exhaustive search against the key space. Poorly chosen encryption algorithms and parameters allow attackers to exhaust the key space through brute force and give them unauthorized access to the data.

T.KNOWN_PLAINTEXT

Threat agents know plaintext in regions of storage devices, especially in uninitialized regions (all zeroes) as well as regions that contain well known software such as operating systems. A poor choice of encryption algorithms, encryption modes, and initialization vectors along with known plaintext could allow an attacker to recover the effective DEK, thus providing unauthorized access to the previously unknown plaintext on the storage device.

T.UNAUTHORIZED_DATA_ACCESS

The cPP addresses the primary threat of unauthorized disclosure of protected data stored on a storage device. If an adversary obtains a lost or stolen storage device (e.g., a storage device contained in a laptop or a portable external storage device), they may attempt to connect a targeted storage device to a host of which they have complete control and have raw access to the storage device (e.g., to specified disk sectors, to specified blocks).

T.UNAUTHORIZED_FIRMWARE_MODIFY

An attacker attempts to modify the firmware of the storage device via a command from the AA or from the OE that may compromise the security features of the TOE.

T.UNAUTHORIZED_UPDATE

Threat agents may attempt to perform an update of the product which compromises the security features of the TOE. Poorly chosen update protocols, signature generation and verification algorithms, and parameters may allow attackers to install software that bypasses the intended security features and provides them unauthorized access to data.

3.2 Assumptions

A.INITIAL_DRIVE_STATE

Users enable Full Drive Encryption on a newly provisioned storage device free of protected data in areas not targeted for encryption. It is also assumed that data intended for protection should not be on the targeted storage media until after provisioning. The cPP does not intend to include requirements to find all the areas on storage devices that potentially contain protected data. In some cases, it may not be possible - for example, data contained in “bad” sectors. While inadvertent exposure to data contained in bad sectors or un-partitioned space is unlikely, one may use forensics tools to recover data from such areas of the storage device. Consequently, the cPP assumes bad sectors, un-partitioned space, and areas that must contain unencrypted code (e.g., MBR and AA/EE pre-authentication software) contain no protected data.

A.PHYSICAL

The platform is assumed to be physically protected in its Operational Environment and not subject to physical attacks that compromise the security or interfere with the platform’s correct operation.

A.PLATFORM_STATE

The platform in which the storage device resides (or an external storage device is connected) is free of malware that could interfere with the correct operation of the product.

A.POWER_DOWN

The user does not leave the platform and/or storage device unattended until all volatile memory is erased after a power-off, so memory remnant attacks are infeasible.

Authorized users do not leave the platform and/or storage device in a mode where sensitive information persists in non-volatile storage (e.g., lock screen). Users power the platform and/or storage device down or place it into a power managed state, such as a “hibernation mode”.

A.STRONG_CRYPTO

All cryptography implemented in the Operational Environment and used by the TOE meets the requirements listed in the cPP. This includes generation of external token authorization factors by an RBG.

A.TRAINED_USER

Users follow the provided guidance for securing the TOE and authorization factors. This includes conformance with authorization factor strength, using external token authentication factors for no other purpose and ensuring external token authorization factors are securely stored separately from the storage device or platform. The user should also be trained on how to power off their system.

A.TRUSTED_CHANNEL

Communication among and between product components (e.g., AA and EE) is sufficiently protected to prevent information disclosure. In cases in which a single product fulfils both cPPs, the communication between the components does not extend beyond the boundary of the TOE (i.e., communication path is within the TOE boundary) so this assumption is inherently met. In cases in which independent products satisfy the requirements of the AA and EE, the physically close proximity of the two products during their operation means that the threat agent has very little opportunity to interpose itself in the channel between the two without the user noticing and taking appropriate actions.

3.3 Organizational Security Policies

This PP defines no Organizational Security Policies.

4 Security Objectives

4.1 Security Objectives for the Operational Environment

The Operational Environment (OE) of the TOE implements technical and procedural measures to assist the TOE in correctly providing its security functionality. This part wise solution forms the security objectives for the Operational Environment and consists of a set of statements describing the goals that the Operational Environment should achieve.

OE.INITIAL_DRIVE_STATE

The OE provides a newly provisioned or initialized storage device free of protected data in areas not targeted for encryption.

Rationale: Since the cPP requires all protected data to be encrypted, A.INITIAL_DRIVE_STATE assumes that the initial state of the device targeted FDE is free of protected data in those areas of the drive where encryption will not be invoked (e.g., MBR, AA, or EE pre-authentication software). Given this known start state, the product (once installed and operational) ensures partitions of logical blocks of user accessible data is protected.

OE.PASSWORD_STRENGTH

An authorized user will be responsible for ensuring that the password authorization factor conforms to guidance from the organization that uses or owns the TOE.

Rationale: Users are properly trained [A.TRAINED_USER] to create authorization factors that conform to administrative guidance.

OE.PHYSICAL

The Operational Environment will provide a secure physical computing space such than an adversary is not able to make modifications to the environment or to the TOE itself.

Rationale: As stated in section 1.6, the use case for this cPP is to protect data-at-rest on a device where the adversary receives it in a powered off state and has no prior access.

OE.PLATFORM_STATE

The platform in which the storage device resides (or an external storage device is connected) is free of malware that could interfere with the correct operation of the product.

Rationale: A platform free of malware [A.PLATFORM_STATE] prevents an attack vector that could potentially interfere with the correct operation of the product.

OE.POWER_DOWN

Volatile memory is erased after entering a compliant power-saving state or turned off so memory remnant attacks are infeasible.

Rationale: Users are properly trained [A.TRAINED_USER] to not leave the storage device unattended until it is in a compliant power-saving state or fully turned off.

OE.SINGLE_USE_ET

External tokens that contain authorization factors will be used for no other purpose than to store the external token authorization factor.

Rationale: Users are properly trained [A.TRAINED_USER] to use external token authorization factors as intended and for no other purpose.

OE.STRONG_ENVIRONMENT_CRYPTO

The Operational Environment will provide a cryptographic function capability that is commensurate with the requirements and capabilities of the TOE and Appendix A.

Rationale: All cryptography implemented in the Operational Environment and used by the product meets the requirements listed in this cPP [A.STRONG_CRYPTO].

OE.TRAINED_USERS

Authorized users will be properly trained and follow all guidance for securing the TOE and authorization factors.

Rationale: Users are properly trained [A.TRAINED_USER] to create authorization factors that conform to guidance, not store external token authorization factors with the device, and power down the TOE when required [OE.PLATFORM_STATE].

OE.TRUSTED_CHANNEL

Communication among and between product components (i.e., AA and EE) is sufficiently protected to prevent information disclosure.

Rationale: In situations where there is an opportunity for an adversary to interpose themselves in the channel between the AA and the EE, a trusted channel should be established to prevent exploitation. [A.TRUSTED_CHANNEL] assumes the existence of a trusted channel between the AA and EE, except for when the boundary is within and does not breach the TOE or is in such close proximity that a breach is not possible without detection.

4.2 Security Objectives Rationale

This section describes how the assumptions and organizational security policies map to operational environment security objectives.

Table 2: Security Objectives Rationale

Assumption or OSP	Security Objectives	Rationale
A.INITIAL_DRIVE_STATE	OE.INITIAL_DRIVE_STATE	The operational environment objective OE.INITIAL_DRIVE_STATE is realized through A.INITIAL_DRIVE_STATE.
A.PHYSICAL	OE.PHYSICAL	The operational environment objective OE.PHYSICAL is realized through A.PHYSICAL.
A.PLATFORM_STATE	OE.PLATFORM_STATE	The operational environment objective OE.PLATFORM_STATE is realized through A.PLATFORM_STATE.
A.POWER_DOWN	OE.POWER_DOWN	The operational environment objective OE.POWER_DOWN is realized through A.POWER_DOWN.
A.STRONG_CRYPTO	OE.STRONG_ENVIRONMENT_CRYPTO	The operational environment objective OE.STRONG_ENVIRONMENT_CRYPTO is realized through A.STRONG_CRYPTO.
A.TRAINED_USER	OE.PASSWORD_STRENGTH	The operational environment objective OE.PASSWORD_STRENGTH is realized through A.TRAINED_USER.

	OE.POWER_DOWN	The operational environment objective OE.POWER_DOWN is realized through A.TRAINED_USER.
	OE.SINGLE_USE_ET	The operational environment objective OE.SINGLE_USE_ET is realized through A.TRAINED_USER.
	OE.TRAINED_USERS	The operational environment objective OE.TRAINED_USERS is realized through A.TRAINED_USER.
A.TRUSTED_ CHANNEL	OE.TRUSTED_ CHANNEL	The operational environment objective OE.TRUSTED_CHANNEL is realized through A.TRUSTED_CHANNEL.

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 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

The individual security functional requirements are specified in the sections below.

5.1.1 Cryptographic Support (FCS)

FCS_CKM.1/DEK Cryptographic Key Generation (Data Encryption Key)

FCS_CKM.1.1/DEK

The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation ~~algorithm~~ **method** [**selection**: *generate a DEK using the RBG as specified in FCS_CKM.1/SKG, accept a DEK that is generated by the RBG provided by the OE, accept a DEK that is wrapped as specified in FCS_COP.1/KeyWrap*] and specified cryptographic key sizes [256 bits] ~~that meet the following: [**assignment**: *list of standards*]~~.

Application Note: This SFR is iterated because additional iterations are defined as optional requirements in Appendix A. This iteration was chosen specifically to ensure consistency between the FDE cPPs.

The purpose of this requirement is to explain DEK generation during provisioning.

If the TOE can be configured to obtain a DEK through more than one method, the ST author chooses the applicable options within the selection. For example, the TOE may generate random numbers with an approved RBG to create a DEK, as well as provide an interface to accept a DEK from the environment.

If the ST author chooses the first or third option, or both in the selection, the corresponding requirement is pulled from Appendix A and included in the body of the ST.

Evaluation Activities

FCS_CKM.1/DEK

TSS

The evaluator shall examine the TSS to determine that it describes how the TOE obtains a DEK (either generating the DEK or receiving from the environment).

If the TOE generates a DEK, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG.1 is invoked. If the DEK is generated outside of the TOE, the evaluator checks to ensure that for each platform identified in the TOE the TSS, it describes the interface used by the TOE to invoke this functionality. The evaluator uses the description of the interface between the RBG and the TOE to determine that it requests a key greater than or equal to the required key sizes.

If the TOE received the DEK from outside the OE, then the evaluator shall examine the TSS to determine that the DEK is sent wrapped using the appropriate encryption algorithm.

Guidance

There is no AGD for this activity.

KMD

If the TOE received the DEK from outside the OE, then the evaluator shall verify that the KMD describes how the TOE unwraps the DEK.

Tests

- *Test FCS_CKM.1/DEK:1: The evaluator shall configure the TOE to ensure the functionality of all selections.*

FCS_CKM.6/Power Cryptographic Key and Key Material Destruction (Power Management)

FCS_CKM.6.1/Power

The TSF shall destroy [*all key material, BEV, and authentication factors stored in plaintext*] when [*transitioning to a compliant power saving state as defined by FPT_PWR_EXT.1*].

Application Note: The TOE may end up in a non-compliant power saving state indistinguishable from a compliant power state (e.g. as result of sudden or unexpected power loss). Guidance documentation must state what conditions result in clear text keys or key materials to stay in volatile memory and identify mitigation measures that result in erasure of volatile memory.

FCS_CKM.6.2/Power

The TSF shall destroy cryptographic keys and keying material specified by FCS_CKM.6.1/Power in accordance with a specified cryptographic key destruction method **[selection: *instruct the operational environment to erase*, *erase*]** that meets the following: [*a key destruction method specified in FCS_CKM_EXT.6*].

Application Note: In some cases, erasure of keys from volatile memory is only supported by the operational environment, in which case the operational environment must expose a well-documented mechanism or interface to invoke the memory erase operation.

Self-encrypting drives do not store keys in the Operational Environment and cannot

instruct the Operational Environment to perform functionality so they are not expected to select “instruct the Operational Environment to clear”.

Evaluation Activities

FCS_CKM.6.1/Power

TSS

The evaluator shall verify the TSS provides a description of what keys and key material are destroyed when entering any compliant power saving state.

Guidance

The evaluator shall validate that guidance documentation contains clear warnings and information on conditions in which the TOE may end up in a non-compliant power saving state indistinguishable from a compliant power saving state. In that case it must contain mitigation instructions on what to do in such scenarios.

KMD

The evaluator shall verify the KMD includes a description of the areas where keys and key material reside.

The evaluator shall verify the KMD includes a key lifecycle that includes a description where key material resides, how the key material is used, and how the material is destroyed once it is not needed and that the documentation in the KMD follows FCS_CKM_EXT.6 for the destruction.

Tests

There are no test evaluation activities for this SFR.

FCS_CKM.6.2/Power

TSS

The evaluator shall verify the TSS provides a high level description of how keys stored in volatile memory are destroyed. The valuator to verify that TSS outlines:

- *if and when the TSF or the Operational Environment is used to destroy keys from volatile memory;*
- *if and how memory locations for (temporary) keys are tracked;*
- *details of the interface used for key erasure when relying on the OE for memory erasure.*

Guidance

The evaluator shall check the guidance documentation if the TOE depends on the Operational Environment for memory erasure and how that is achieved.

KMD

The evaluator shall check to ensure the KMD lists each type of key, its origin, possible memory locations in volatile memory.

Tests

There are no test evaluation activities for this SFR.

FCS_CKM_EXT.6 Cryptographic Key Destruction Types

FCS_CKM_EXT.6.1

The TSF shall use [**selection**: *FCS_CKM.6/GENHW*, *FCS_CKM.6/SW*, *FCS_CKM.6/TOEHW*] and [**selection**: *FCS_CKM.6/KEK*, *no other method*] key destruction methods.

Application Note: The specific key destruction methods that will be claimed are dependent on the selections made for where keys are stored.

If multiple selections are made, the TSS should identify which keys are destroyed according to which selections.

Evaluation Activities

FCS_CKM_EXT.6
TSS
(Key Management Description may be used if necessary details describe proprietary information)

The evaluator shall examine the TOE's keychain in the TSS/KMD and verify all keys subject to destruction are destroyed according to one of the specified methods.

Guidance
There is no AGD for this activity.

Tests
There is no test for this activity.

FCS_COP.1/Hash Cryptographic Operation (Hash Algorithm)

FCS_COP.1.1/Hash

The TSF shall perform [*cryptographic hashing*] in accordance with a specified cryptographic algorithm [**selection**: *SHA-256, SHA-384, SHA-512, SHA3-384, SHA3-512*] that meet the following: [**selection**: *ISO/IEC 10118-3:2018 [SHA, SHA3], FIPS PUB 180-4 [SHA], FIPS PUB 202 [SHA3]*].

Application Note:

- SHA3 hashes may be used for internal hardware functionality such as boot integrity checks and DRBGs, and
- SHA-256 is permitted only for use as a PRF, including as part of a key derivation function or RBG, or as part of LMS or XMSS.

The hash selection should be consistent with the overall strength of the algorithm used for signature generation. For example, the TOE should choose SHA-384 for 3072-bit RSA, 4096-bit RSA, or ECC with P-384; and SHA-512 for ECC with P-521.

Evaluation Activities

FCS_COP.1/Hash
TSS
The evaluator shall examine the TSS to verify that if SHA-256 is selected, that it is being used only as a PRF, including as part of a key derivation function or RBG, or as part of LMS or XMSS, and not as a hash algorithm.

Guidance
There are no AGD evaluation activities for this SFR.

KMD

There are no KMD evaluation activities for this SFR.

Tests

The following tests may 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.

SHA-256, SHA-384, SHA-512

To test the TOE's ability to generate hash digests using SHA2 the evaluator shall perform the Algorithm Functional Test, Monte Carlo Test, and Large Data Test for each claimed SHA2 algorithm.

Algorithm Functional Test

The evaluator shall generate a number of test cases equal to the block size of the hash (512 for SHA2-256; 1024 for the other SHA2 algorithms).

Each test case is to consist of random data of a random length between 0 and 65536 bits, or the largest size supported.

Each test case is to consist of random data of a random length between 0 and 65536 bits, or the largest size supported.

Monte Carlo Test

Monte Carlo tests begin with a single seed and run 100 iterations of the chained computation.

There are two versions of the Monte Carlo test for SHA-1 and SHA-2. Either one is acceptable. For the Standard Monte Carlo test the message hashed is always three times the length of the initial seed.

```
For j = 0 to 99
        A = B = C = SEED
        For i = 0 to 999
                MSG = A || B || C
                MD = SHA(MSG)
                A = B
                B = C
                C = MD
        Output MD
        SEED = MD
```

For the alternate version of the Monte Carlo Test, the hashed message is always the same length as the seed.

```
INITIAL_SEED_LENGTH = LEN(SEED)
For j = 0 to 99
        A = B = C = SEED
        For i = 0 to 999
                MSG = A || B || C
                if LEN(MSG) >= INITIAL_SEED_LENGTH:
                        MSG = leftmost INITIAL_SEED_LENGTH bits of MSG
                else:
                        MSG = MSG || INITIAL_SEED_LENGTH - LEN(MSG) 0 bits
                MD = SHA(MSG)
                A = B
                B = C
                C = MD
```

```
        Output MD
        SEED = MD
```

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

Large Data Test

The implementation must be tested against one test case each on large data messages of 1GB, 2GB, 4GB, and 8GB of data as supported. The data need not be random. It may, for example, consist of a repeated pattern of 64 bits.

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

SHA3-384, SHA3-512 *To test the TOE's ability to generate hash digests using SHA3 the evaluator shall perform the Algorithm Functional Test, Monte Carlo Test, and Large Data Tests for each claimed SHA3 algorithm.*

Algorithm Functional Test

Generate a test case consisting of random data for every message length from 0 bits (or the smallest supported message size) to rate bits, where rate equals

- *832 for SHA3-384 and*
- *576 for SHA3-512.*

Additionally, generate tests cases of random data for messages of every multiple of (rate+1) bits starting at length rate, and continuing until 65535 is exceeded.

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

Monte Carlo Test

Monte Carlo tests begin with a single seed and run 100 iterations of the chained computation.

For this Monte Carlo Test, the hashed message is always the same length as the seed.

```
MD[0] = SEED
INITIAL_SEED_LENGTH = LEN(SEED)
For 100 iterations
        For i = 1 to 1000
                MSG = MD[i-1];
                if LEN(MSG) >= INITIAL_SEED_LENGTH:
                        MSG = leftmost INITIAL_SEED_LENGTH bits of MSG
                else:
                        MSG = MSG || INITIAL_SEED_LENGTH - LEN(MSG) 0 bits
                MD[i] = SHA3(MSG)
        MD[0] = MD[1000]
        Output MD[0]
```

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

Large Data Test

The implementation must be tested against one test case each on large data messages of 1GB, 2GB, 4GB, and 8GB of data as supported. The data need not be random. It may, for example, consist of a repeated pattern of 64 bits.

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

FCS_COP.1/SigVer Cryptographic Operation - Signature Verification

FCS_COP.1.1/SigVer

The TSF shall perform [digital signature verification] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 3 provides the allowable choices for completion of the selection operations of FCS_COP.1/SigVer.

Table 3: Allowable choices for FCS_COP.1/SigVer

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
RSA-PKCS	RSASSA-PKCS1-v1_5	Modulus of size [**selection**: *3072, 4096, 6144, 8192*] bits and hash[**selection**: *SHA-384, SHA-512*]	RFC 8017 (Section 8.2) [PKCS #1 v2.2] FIPS PUB 186-5 (Section 5.4) [RSASSA-PKCS1-v1_5]
RSA-PSS	RSASSA-PSS	Modulus of size [**selection**: *3072, 4096, 6144, 8192*] bits and hash[**selection**: *SHA-384, SHA-512*]	RFC 8017 (Section 8.1) [PKCS#1 v2.2] FIPS PUB 186-5 (Section 5.4) [RSASSA-PSS]
ECDSA	ECDSA	Elliptic Curve [**selection**: *P-384, P-521*] using hash [**selection**: *SHA-384, SHA-512*]	[**selection**: *ISO/IEC 14888-3:2018 (Subclause 6.6), FIPS PUB 186-5 (Section 6.4.2)*] [ECDSA] NIST SP-800 186 (Section 4) [NIST Curves]
LMS	LMS	Private key size = [**selection**: • *192 bits with [**selection**: SHA-256/192, SHAKE256/192]* • *256 bits with [**selection**: SHA-256, SHAKE256]*]	RFC 8554 [LMS] NIST SP 800-208 [parameters]

		Winternitz parameter = [**selection**: *1*, *2*, *4*, *8*] Tree height = [**selection**: *5*, *10*, *15*, *20*, *25*]	
XMSS	XMSS	Private key size = [**selection**: • *192 bits with [**selection**: SHA-256/192, SHAKE256/192]* • *256 bits with [**selection**: SHA-256, SHAKE256]*] Tree height = [**selection**: *10*, *16*, *20*]	RFC 8391 [XMSS] NIST SP 800-208 [parameters]
ML-DSA	ML-DSA Signature Verification	Parameter set = ML-DSA-87	NIST FIPS 204 (Section 5.3)

Application Note: The ST Author should choose the algorithm implemented to perform verification of digital signatures. 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.

Evaluation Activities

FCS_COP.1/SigVer

TSS

The evaluator shall examine the TSS to verify that any one-time values such as nonces or masks are constructed and used in accordance with the relevant standards.

Guidance

There are no guidance activities for this SFR.

KMD

There are no KMD evaluation activities for this SFR.

Tests

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.

RSA-PKCS Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
RSA-PKCS	RSASSA-PKCS1-v1_5	Modulus of size [**selection:** 3072, 4096, 6144, 8192] bits, hash [**selection:** SHA-384, SHA-512]	RFC 8017 (Section 8.2) [PKCS #1 v2.2] NIST FIPS PUB 186-5 (Section 5.4) [RSASSA-PKCS1-v1_5]

To test the TOE's ability to perform RSA Digital Signature Verification using PKCS1-v1_5 signature type, the evaluator shall perform Generated Data Test using the following input parameters:

- *Modulus size [3072, 4096, 6144, 8192] bits*
- *Hash algorithm [SHA-384, SHA-512]*

Generated Data Test

For each supported combination of the above parameters, the evaluator shall cause the TOE to generate six test cases using a random message and its signature such that the test cases are modified as follows:

- *One test case is left unmodified*
- *For one test case the Message is modified*
- *For one test case the Signature is modified*
- *For one test case the exponent (e) is modified*
- *For one test case the IR is moved*
- *For one test case the Trailer is moved*

The TOE must correctly verify the unmodified signatures and fail to verify the modified signatures.

RSA-PSS Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
RSA-PSS	RSASSA-PSS	Modulus of size [**selection:** 3072, 4096, 6144, 8192] bits, hash [**selection:** SHA-384, SHA-512]	RFC 8017 (Section 8.2) [PKCS #1 v2.2] NIST FIPS PUB 186-5 (Section 5.4) [RSASSA-PSS]

To test the TOE's ability to perform RSA Digital Signature Verification using PSS signature type, the evaluator shall perform the Generated Data Test using the following input parameters:

- *Modulus size [3072, 4096, 6144, 8192] bits*
- *Hash algorithm [SHA-384, SHA-512]*
- *Salt length [0-hash length]*
- *Mask function [MGF1]*

Generated Data Test

For each supported combination of the above parameters, the evaluator shall cause the TOE to generate six test cases using random data such that the test cases are modified as follows:

- *One test case is left unmodified*
- *For one test case the Message is modified*
- *For one test case the Signature is modified*
- *For one test case the exponent (e) is modified*
- *For one test case the IR is moved*
- *For one test case the Trailer is moved*

The TOE must correctly verify the unmodified signatures and fail to verify the modified signatures.

DSA Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
DSA	DSA	Domain parameters for (L, N) = [(3072, 256)] bits	FIPS PUB 186-4 (Section 4.7) [DSA Signature Verification]

To test the TOE's ability to perform DSA Digital Signature Verification, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *(L, N) = (3072, 256)*
- *Hash algorithm [SHA-384, SHA-512]*

Algorithm Functional Test

For each supported combination of the above parameters, the evaluator shall cause the TOE to generate 15 test cases consisting of messages and signatures such that the 15 test cases are modified as follows:

- *Three test cases are left unmodified*
- *For three test cases the Message is modified*
- *For three test cases the key is modified*
- *For three test cases the r value is modified*
- *For three test cases the s value is modified*

The TOE must correctly verify the unmodified signatures and fail to verify the modified signatures.

ECDSA Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
ECDSA	ECDSA	Elliptic Curve [**selection:** P-384, P-521] and hash function using [**selection:** SHA-384, SHA-512]	[**selection:** ISO/IEC 14888-3:2018 (Subclause 6.6), NIST FIPS PUB 186-5 (Sections 6.3.1, 6.4.1] [ECDSA] NIST SP-800 186 (Section 4) [NIST Curves]

To test the TOE's ability to perform ECDSA Digital Signature Verification, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *Elliptic Curve [P-384, P-521]*
- *Hash algorithm [SHA-384, SHA-512]*

Algorithm Functional Test

For each supported combination of the above parameters, the evaluator shall cause the TOE to generate test cases consisting of messages and signatures such that the 21 test cases are modified as follows:

- *Three test cases are left unmodified*
- *For three test cases the Message is modified*
- *For three test cases the key is modified*
- *For three test cases the r value is modified*
- *For three test cases the s value is modified*
- *For three test cases the value r is zeroed*
- *For three test cases the value s is zeroed*

The TOE must correctly verify the unmodified signatures and fail to verify the modified signatures.

LMS Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
LMS	LMS	Private key size = [**selection:** 192 bits with [**selection:** SHA256/192, SHAKE256/192], 256 bits with [**selection:** SHA-256, SHAKE256]], Winternitz parameter = [**selection:** 1, 2, 4, 8], and tree height = [**selection:** 5, 10, 15, 20, 25]	RFC 8554 [LMS] NIST SP 800-208 [parameters]

To test the TOE's ability to verify cryptographic digital signature using LMS, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *Hash algorithm [SHA-256/192, SHAKE256/192, SHA-256, SHAKE256]*
- *Winternitz [1, 2, 4, 8]*
- *Tree height [5, 10, 15, 20, 25]*

Algorithm Functional Test

For each supported combination of the above parameters, the evaluator shall generate 4 test cases consisting of signed messages and keys, such that

- *One test case is unmodified (i.e. correct)*
- *For one test case modify the message, i.e. the message is different*
- *For one test case modify the signature, i.e. signature is different*
- *For one test case modify the signature header so that it is a valid header for a different LMS parameter set.*

The TOE must correctly verify the unmodified test case and fail to verify the modified test cases.

XMSS Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
XMSS	XMSS	Private key size = [**selection:** 192 bits with [**selection:** SHA256/192, SHAKE256/192], 256 bits with [**selection:** SHA-256, SHAKE256]], and tree height = [**selection:** 10, 16, 20]	RFC 8391 [XMSS] NIST SP 800-208 [parameters]

To test the TOE's ability to verify digital signatures using XMSS or XMSS MT, the evaluator shall perform the XMSS digital signature verification test using the following input parameters:

- *Hash algorithm [SHA-256/192, SHAKE256/192, SHA-256, SHAKE256]*
- *Tree height [10, 16, 20]*

XMSS Digital Signature Verification Test

For each supported combination of the above parameters, the evaluator shall generate four test cases consisting of signed messages and keys, such that

- *One test case is unmodified (i.e. correct)*
- *For one test case modify the message, i.e. the message is different*
- *For one test case modify the signature, i.e. signature is different*
- *For one test case modify the signature header so that it is a valid header for a different XMSS parameter set*

The evaluator shall verify the correctness of the implementation by verifying that the TOE correctly verifies the unmodified test case and fails to verify the modified test cases.

ML-DSA Signature Verification

Identifier	Cryptographic Algorithm Parameters	Cryptographic Key Sizes	List of Standards
ML-DSA	ML-DSA SigVer	Parameter set = ML-DSA-87	NIST FIPS PUB 204 (Section 5.2)

To test the TOE's ability to validate digital signatures using ML-DSA, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *Parameter set [ML-DSA-87]*
- *Previously generated signed Message [8-65535] bytes*
- *Mu value (if generated externally)*
- *Context (for external interface testing)*
- *Previously generated Public key (pk)*
- *Previously generated Signature*

Algorithm Functional Test

For each combination of supported parameter set and capabilities, the evaluator shall require the implementation under test to validate 15 signatures. Each group of 15 test cases is modified as follows:

- *Three test cases are left unmodified*
- *For three test cases the Signed message is modified*
- *For three test cases the component of the signature that commits the signer to the message is modified*

- *For three test cases the component of the signature that allows the verifier to construct the vector z is modified*
- *For three test cases the component of the signature that allows the verifier to construct the hint array is modified*

The TOE must correctly verify the unmodified signatures and fail to verify the modified signatures.

FCS_COP.1/SKC Cryptographic Operation (AES Data Encryption/Decryption)

FCS_COP.1.1/SKC

The TSF shall perform [symmetric-key data encryption/decryption] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 4 provides the allowed choices for completion of the selection operations of FCS_COP.1/SKC.

Table 4: Allowed choices for FCS_COP.1/SKC

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
AES-CBC	AES in CBC mode with non-repeating and unpredictable IVs	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 10116:2017 (Clause 7), NIST SP 800-38A*] [CBC]
AES-XTS	AES in XTS mode with unique tweak values that are consecutive non-negative integers starting at an arbitrary non-negative integer	512 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] [**selection**: *IEEE Std. 1619-2018, NIST SP 800-38E*] [XTS]
AES-GCM	AES in GCM mode with non-repeating IVs using [**selection**: *deterministic, RBG-based*], IV construction; the tag must be of length [**selection**: *96, 104, 112, 120, 128*] bits.	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES]

[**selection**: *ISO/IEC 19772:2020 (Clause 10), NIST SP 800-38D*] [GCM]

Application Note: The intent of this requirement in the context of this cPP is to provide an SFR that expresses the appropriate symmetric encryption/decryption algorithms suitable for use in the TOE. If the ST author incorporates the validation requirement (FCS_VAL_EXT.1) and chooses to select the option to decrypt a known value and perform a comparison, this is the requirement used to specify the algorithm, modes, and key sizes the ST author can choose from.

Evaluation Activities

FCS_COP.1/SKC

TSS

The evaluator shall examine the TSS to ensure that it describes the construction of any IVs, tweak values, and counters in conformance with the relevant specifications.

If AES-XTS is claimed then the evaluator shall examine the TSS to verify that the TOE creates full-length keys by methods that ensure that the two key halves are different and independent. If a GCM mode algorithm is selected, then the evaluator shall examine the TOE summary specification to confirm 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. The evaluator shall also confirm that for each invocation of GCM, the length of the plaintext is at most (2^{32})-2 blocks.

Guidance

There is no AGD for this activity.

KMD

There is no KMD for this activity.

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.

AES-CBC

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-CBC	AES in CBC mode with non-repeating and unpredictable	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS

	IVs		PUB 197] [AES] [**selection:** ISO/IEC 10116:2017 (Clause 7), NIST SP 800-38A] [CBC]

To test the TOE's ability to encrypt/decrypt data using AES in CBC mode, the evaluator shall perform Algorithm Functional Tests and Monte Carlo Tests using the following input parameters:

- *Key size [256] bits*
- *Direction [encryption, decryption]*

Algorithm Functional Tests

Algorithm Functional Tests are designed to verify the correct operation of the logical components of the algorithm implementation under normal operation using different block sizes. For AES-CBC, there are two types of AFTs:

Known-Answer Tests

For each combination of direction and claimed key size, the TOE must be tested using the GFSBox, KeySbox, VarTxt, and VarKey test cases listed in Appendixes B through E of The Advanced Encryption Standard Algorithm Validation Suite (AESAVS), NIST, 15 November 2002.

Multi-Block Message Tests

For each combination of direction and claimed key size, the TOE must be tested against 10 test cases consisting of a random IV, random key, and random plaintext/ciphertext. The plaintext/ciphertext starts with a length of 16 bytes and increases by 16 bytes for each test case until reaching 160 bytes.

Monte Carlo Tests

Monte Carlo tests are intended to test the implementation under strenuous conditions. The TOE must process the test cases according to the following algorithm once for each combination of direction and key size:

```
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]f
                AES_KEY_SHUFFLE(Key, CT)
                IV[i+1] = CT[j]
                PT[0] = CT[j-1]
}
```

where

```
AES_KEY_SHUFFLE
```

is defined as:

```
If ( keylen = 128 )
        Key[i+1] = Key[i] xor MSB(CT[j], 128)
```

```
If ( keylen = 192 )
        Key[i+1] = Key[i] xor (LSB(CT[j-1], 64) || MSB(CT[j], 128))
If ( keylen = 256 )
        Key[i+1] = Key[i] xor (MSB(CT[j-1], 128) || MSB(CT[j], 128))
```

The above pseudocode is for encryption. For decryption, swap all instances of CT and PT.

The initial IV, key, and plaintext/ciphertext should be random.

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-XTS

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-XTS	AES in XTS mode with unique tweak values that are consecutive non-negative integers starting at an arbitrary non-negative integer	512 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] [**selection:** IEEE Std. 1619-2018, NIST SP 800-38E] [XTS]

To test the TOE's ability to encrypt/decrypt data using AES in XTS mode, the evaluator shall perform the Single Data Unit Test and the Multiple Data Unit Test using the following input parameters:

- *Direction [encryption, decryption]*
- *Key size [512] bits*
- *Tweak value format [128-bit hex string, data unit sequence number]*

Single Data Unit Test

For each combination of claimed key size, direction, and supported tweak value format, the evaluator shall generate 50 test cases consisting of random payload data. The payload data size is determined randomly for each test case from supported values within the range [128-65536] bits. The payload size and data unit size must be equal.

Multiple Data Unit Test

For each combination of claimed key size, direction, and supported tweak value format, the evaluator shall generate 50 test cases consisting of random payload data. The payload data size is determined randomly for each test case from supported values within the range [128-65536] bits. Likewise, the data unit size is determined randomly for each test case from supported values within the range [128-65535] bits. The payload size and data unit size must not be equal.

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.

AES-GCM

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards

AES-GCM	AES in GCM mode with non-repeating IVs using [**selection:** deterministic, RBG-based] IV construction; the tag must be of length [**selection:** 96, 104, 112, 120, or 128] bits.	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] [**selection:** ISO/IEC 19772:2020 (Clause 10), NIST SP 800-38D] [GCM]

To test the TOE's implementation of AES-GCM authenticated encryption functionality the evaluator shall perform the Encryption Algorithm Functional Tests and Decryption Algorithm Functional Tests as described below using the following input parameters:

- *Key Size [256] bits*
- *Associated data size [0-65536] bits*
- *Payload size [0-65536] bits*
- *IV size [96] bits*
- *Tag size [96, 104, 112, 120, 128] bits*

Encryption Algorithm Functional Tests

The evaluator shall generate 15 test cases using random data for each combination of the above parameters as follows:

- *Each claimed key size,*
- *Each supported tag size,*
- *Four supported non-zero payload sizes, such that two are multiples of 128 bits and two are not multiples of 128 bits,*
- *Four supported non-zero associated data sizes, such that two are multiples of 128 bits and two are not multiples of 128 bits, and*
- *An associated data size of zero, if supported.*

Note that the IV size is always 96 bits.

The evaluator shall compare the output from each test case against results generated by a known- good implementation with the same input parameters.

Decryption Algorithm Functional Tests

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 test cases for the supported combinations of the parameters as described above. 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.

FCS_KYC_EXT.2 Key Chaining (Recipient)

FCS_KYC_EXT.2.1

The TSF shall accept a BEV of at least 256 bits.

FCS_KYC_EXT.2.2

The TSF shall maintain a chain of intermediary keys originating from the BEV to the DEK using the following methods: [**selection**:

- *key encryption as specified in FCS_COP.1/KeyEnc*
- *key transport as specified in FCS_COP.1/KeyEncap*
- *key wrapping as specified in FCS_COP.1/KeyWrap*

- *key derivation as specified in FCS_CKM.5*
- *key combining as specified in FCS_SMC_EXT.1*

] while maintaining an effective strength of [256 bits] for symmetric keys and an effective strength of [**selection**: *not applicable*, *128 bits*, *192 bits*, *256 bits*] for asymmetric keys.

Application Note: Key Chaining is the method of using multiple layers of encryption keys to ultimately secure the protected data encrypted on the drive. The number of intermediate keys will vary – from two (e.g., using the BEV as an intermediary key to wrap the DEK) to many. This applies to all keys that contribute to the ultimate wrapping or derivation of the DEK; including those in areas of protected storage (e.g. TPM stored keys, comparison values).

The BEV is considered to be equivalent to keying material and therefore additional checksums or similar values are not the BEV, even if they are sent with the BEV.

Once the ST author has selected a method to create the chain (either by deriving keys or unwrapping them), they pull the appropriate requirement out of Appendix B. It is allowable for an implementation to use both methods.

The method the TOE uses to chain keys and manage/protect them is described in the Key Management Description; see Appendix E for more information.

Evaluation Activities

FCS_KYC_EXT.2
TSS
There is no TSS for this activity

Guidance
There is no AGD for this activity.

KMD
The evaluator shall examine the KMD to ensure it describes a high level key hierarchy and details of the key chain. The description of the key chain shall be reviewed to ensure it maintains a chain of keys using key wrap or key derivation methods that meet FCS_CKM.5, FCS_COP.1/KeyWrap, FCS_COP.1/KeyEncap, and/or FCS_COP.1/KeyEnc.

The evaluator shall examine the KMD to ensure that it describes how the key chain process functions, such that it does not expose any material that might compromise any key in the chain. (e.g. using a key directly as a compare value against a TPM) This description must include a diagram illustrating the key hierarchy implemented and detail where all keys and keying material is stored or what it is derived from. The evaluator shall examine the key hierarchy to ensure that at no point the chain could be broken without a cryptographic exhaust or knowledge of the BEV and the effective strength of the DEK is maintained throughout the Key Chain.

The evaluator shall verify the KMD includes a description of the strength of keys throughout the key chain.

Tests
There is no test for this activity.

FCS_SNI_EXT.1 Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation)

FCS_SNI_EXT.1.1

The TSF shall [**selection**:

- *use no salts*
- *use salts that are generated by a [**selection**: DRBG as specified in FCS_RBG.1, DRBG provided by the OE]*

].

FCS_SNI_EXT.1.2

The TSF shall use [**selection**: *no nonces, unique nonces with a minimum size of [64] bits*].

FCS_SNI_EXT.1.3

The TSF shall [**selection**:

- *use no IVs*
- *create IVs in the following manner [**selection**:*
  - *CBC: IVs shall be non-repeating and unpredictable*
  - *CCM: Nonce shall be non-repeating and unpredictable*
  - *XTS: No IV. Tweak values shall be non-negative integers, assigned consecutively, and starting at an arbitrary non-negative integer;*
  - *GCM: IV shall be non-repeating. The number of invocations of GCM shall not exceed 2^32 for a given secret key*

  ]

].

Application Note: This requirement covers several important factors – the salt must be random, but the nonces only have to be unique. FCS_SNI_EXT.1.3 specifies how the IV should be handled for each encryption mode. CBC, XTS, and GCM are allowed for AES encryption of the data. AES-CCM is an allowed mode for Key Wrapping.

If the TSF uses salts in support of cryptographic operations, and these salts are generated by the TSF, then FCS_RBG.1 must be claimed.

Evaluation Activities

FCS_SNI_EXT.1

TSS

If salts are used, the evaluator shall ensure the TSS describes how salts are generated. The evaluator shall confirm that the salt is generating using an RBG described in FCS_RBG.1 or by the Operational Environment. If external function is used for this purpose, the TSS should include the specific API that is called with inputs.

If IVs or nonces are used, the evaluator shall ensure the TSS describes how nonces are created uniquely and how IVs and tweaks are handled (based on the AES mode). The evaluator shall confirm that the nonces are unique and the IVs and tweaks meet the stated requirements.

Guidance

There is no AGD for this activity.

KMD

There is no KMD for this activity.

Tests
There is no test for this activity

FCS_VAL_EXT.1 Validation

FCS_VAL_EXT.1.1

The TSF shall perform validation of the [*BEV*] using the following methods: [**selection**:

- *key wrap as specified in FCS_COP.1/KeyWrap;*
- *hash the [BEV] as specified in [**selection**: FCS_COP.1/Hash, FCS_COP.1/KeyedHash] and compare it to a stored hashed [**value**];*
- *decrypt a known value using the [**selection**: intermediate key, BEV] specified in FCS_COP.1/SKC and compare it against a stored known value*

].

FCS_VAL_EXT.1.2

The TSF shall require validation of the [*BEV*] prior to [*allowing access to TSF data after exiting a Compliant power saving state*].

Application Note: This SFR is claimed when the TSF validates an authentication factor as a prerequisite to unlocking the key chain as defined in FCS_KYC_EXT.2.

FCS_VAL_EXT.1.3

The TSF shall [**selection**:

- *perform a key sanitization of the DEK upon a [**selection**: configurable number, [**assignment**: ST author specified number]] of consecutive failed validation attempts*
- *institute a delay such that only [**assignment**: ST author specified number of attempts] can be made within a 24 hour period*
- *block validation after [**assignment**: ST author specified number of attempts] of consecutive failed validation attempts*
- *require power cycle or reset of the TOE after [**assignment**: ST author specified number of attempts] of consecutive failed validation attempts*

].

Application Note: "Validation" of the BEV can occur at any point in the key chain, including when the DEK is decrypted. For the purposes of this requirement, validating a key derived from the BEV equates to “validating” the BEV. The purpose of performing secure validation is to not expose any material that might compromise the submask(s).

The TOE validates the BEV prior to processing (e.g. decryption, derivation) of the keychain. When the key wrap in FCS_COP.1/KeyWrap is used, the validation is performed inherently.

The delay must be enforced by the TOE, but this requirement is not intended to address attacks that bypass the product (e.g. attacker obtains hash value or “known” crypto value and mounts attacks outside of the TOE, such as a third party password crackers). The cryptographic functions (i.e., hash, decryption) performed are those specified in FCS_COP.1/Hash, FCS_COP.1/KeyedHash, and FCS_COP.1/SKC.

Evaluation Activities

FCS_VAL_EXT.1.3

TSS

The evaluator shall examine the TSS to determine which authorization factors support validation.

The evaluator shall examine the TSS to review a high-level description if multiple submasks are used within the TOE, how the submasks are validated (e.g., each submask validated before combining, once combined validation takes place).

Guidance

(conditional) If the validation functionality is configurable, the evaluator shall examine the operational guidance to ensure it describes how to configure the TOE to ensure the limits regarding validation attempts can be established.

(conditional) If the validation functionality is specified by the ST author, the evaluator shall examine the operational guidance to ensure that it states the values that the TOE uses for limits regarding validation attempts.

KMD

The evaluator shall examine the KMD to verify that it described the method the TOE employs to limit the number of consecutively failed authorization attempts.

The evaluator shall examine the vendor's KMD to ensure it describes how validation is performed. The description of the validation process in the KMD provides detailed information how the TOE validates the submasks. The KMD describes how the process works, such that it does not expose any material that might compromise the submasks.

Tests

The evaluator shall perform the following tests:

- *Test FCS_VAL_EXT.1.3:1: The evaluator shall determine the limit on the average rate of the number of consecutive failed authorization attempts. The evaluator will test the TOE by entering that number of incorrect authorization factors in consecutive attempts to access the protected data. If the limit mechanism includes any "lockout" period, the time period tested should include at least one such period. Then the evaluator will verify that the TOE behaves as described in the TSS.*
- *Test FCS_VAL_EXT.1.3:2: For each validated authorization factor, ensure that when the user provides an incorrect authorization factor, the TOE prevents the BEV from being forwarded outside the TOE (e.g., to the EE).*

5.1.2 User Data Protection

FDP_DSK_EXT.1 Protection of Data on Disk

FDP_DSK_EXT.1.1

The TSF shall perform full drive encryption in accordance with FCS_COP.1/SKC, such that the drive contains no plaintext protected data.

FDP_DSK_EXT.1.2

The TSF shall encrypt all protected data without user intervention.

Application Note: The intent of this requirement is to specify that encryption of any protected data will not depend on a user electing to protect that data. The drive encryption specified in FDP_DSK_EXT.1 occurs transparently to the user and the decision to protect the data is outside the discretion of the user, which is a characteristic

that distinguishes it from file encryption. The definition of protected data can be found in the glossary.

The cryptographic functions that perform the encryption/decryption of the data may be provided by the Operational Environment (OE). Note that if this is the case, it is assumed that the OE implementation of AES is consistent with the behavior described in FCS_COP.1/SKC.

Evaluation Activities

FDP_DSK_EXT.1

TSS

The evaluator shall examine the TSS to ensure that the description is comprehensive in how the data is written to the disk and the point at which the encryption function is applied. The TSS must make the case that standard methods of accessing the disk drive via the TOEs operating system will pass through these functions.

For the cryptographic functions that are provided by the Operational Environment, the evaluator shall check the TSS to ensure it describes, for each platform identified in the ST, the interfaces used by the TOE to invoke this functionality.

The evaluator shall verify the TSS in performing the evaluation activities for this requirement. The evaluator shall ensure the comprehensiveness of the description, confirms how the TOE writes the data to the disk drive, and the point at which it applies the encryption function.

The evaluator shall verify that the TSS describes the initialization of the TOE and the activities the TOE performs to ensure that it encrypts all the storage devices entirely when a user or administrator first provisions the TOE. The evaluator shall verify the TSS describes areas of the disk that it does not encrypt (e.g., portions associated with the Master Boot Records (MBRs), boot loaders, partition tables, etc.). If the TOE supports multiple disk encryptions, the evaluator shall examine the administration guidance to ensure the initialization procedure encrypts all storage devices on the platform.

Guidance

The evaluator shall review the AGD guidance to determine that it describes the initial steps needed to enable the FDE function, including any necessary preparatory steps. The guidance shall provide instructions that are sufficient, on all platforms, to ensure that all hard drive devices will be encrypted when encryption is enabled.

KMD

The evaluator shall verify the KMD includes a description of the data encryption engine, its components, and details about its implementation (e.g. for hardware: integrated within the device's main SOC or separate co-processor, for software: initialization of the product, drivers, libraries (if applicable), logical interfaces for encryption/decryption, and areas which are not encrypted (e.g. boot loaders, portions associated with the Master Boot Record (MBRs), partition tables, etc.)). The evaluator shall verify the KMD provides a functional (block) diagram showing the main components (such as memories and processors) and the data path between, for hardware, the device's host interface and the device's persistent media storing the data, or for software, the initial steps needed to the activities the TOE performs to ensure it encrypts the storage device entirely when a user or administrator first provisions the product. The hardware encryption diagram shall show the location of the data encryption engine within the data path. The evaluator shall validate that the hardware encryption diagram contains enough detail showing the main components within the data path and that it clearly identifies the data encryption engine.

The evaluator shall verify the KMD provides sufficient instructions for all platforms to ensure that

when the user enables encryption, the product encrypts all hard storage devices. The evaluator shall verify that the KMD describes the data flow from the device's host interface to the device's persistent media storing the data. The evaluator shall verify that the KMD provides information on those conditions in which the data bypasses the data encryption engine (e.g., read-write operations to an unencrypted Master Boot Record area).

The evaluator shall verify that the KMD provides a description of the platform's boot initialization, the encryption initialization process, and at what moment the product enables the encryption. The evaluator shall validate that the product does not allow for the transfer of user data before it fully initializes the encryption.

Tests
The evaluator shall perform the following tests:

- *Test FDP_DSK_EXT.1:1:*
  The evaluator may require the software developer provides special tools which allow inspection of the encrypted drive either in-band or out-of-band.
  - *Step 1 Initialize the TOE: Ensure TOE is initialized and, if hardware, encryption engine is ready*
    - *Provision TOE to encrypt the storage device. Use a known key and the developer tools.*
    - *Determine a random character pattern of at least 64 KB*
    - *Retrieve information on what the device TOE's lowest and highest logical address is for which encryption is enabled.*
  - *Step 2: Write pattern to storage device in multiple locations: Randomly select several logical address locations within the device's lowest to highest address range and write pattern to those addresses*
  - *Step 3: Verify data is encrypted: Evaluator shall access storage using out-of-band tools bypassing encryption engine or inspecting the drive in a locked (secure encrypted) state to verify that data on storage media written to designated locations are not plaintext by comparing them with plaintext pattern.*
  - *Step 4: Re-encrypt data: Evaluator shall enforce the TOE to re-encrypt data using new DEK by either:*
    - *changing the DEK and re-encrypting the data or*
    - *re-provisioning the device with new DEK and repeat step 2 by writing same data to same locations.*
  - *Step 5: Verify data is encrypted using new DEK:*
    - *Evaluator shall access storage using out-of-band tools bypassing encryption engine or inspecting the drive in a locked (secure encrypted) state to verify that data on storage media written to designated locations are not plaintext by comparing them with plaintext pattern.*
    - *Evaluator shall verify that obtained ciphertext is different from the one obtained in Step 3 above. If same ciphertext is obtained, test is considered failing.*

5.1.3 Security Management (FMT)

FMT_SMF.1 Specification of Management Functions

FMT_SMF.1.1

The TSF shall be capable of performing the following management functions: [

a. *change the DEK, as specified in FCS_CKM.1/DEK, when re-provisioning or when commanded*
b. *erase the DEK, as specified in FCS_CKM_EXT.6*
c. *initiate TOE firmware/software updates*
d. *[**selection**: no other functions, configure a password for an update, import a wrapped DEK, configure cryptographic functionality, disable key recovery functionality, securely update the public key needed for trusted update, configure the number of failed validation attempts required to trigger corrective behavior, configure the corrective behavior to issue 1 in the event of an excessive number of failed validation attempts, [**assignment**: other management functions provided by the TSF]]]*

Application Note: The intent of this requirement is to express the management capabilities that the TOE possesses. This means that the TOE must be able to perform the listed functions. "Configure cryptographic functionality" could include key management functions; for example, the BEV will be wrapped or encrypted, and the EE will need to unwrap or decrypt the BEV. In item (d), if no other management functions are provided (or claimed), then "no other functions" should be selected.

For the purposes of this document, key sanitization means to destroy the DEK, using one of the approved destruction methods. This applies to instances of the protected key that exist in non-volatile storage.

Evaluation Activities

FMT_SMF.1
TSS
The evaluator shall ensure the TSS describes how the TOE changes the DEK.

The evaluator shall ensure the TSS describes how the TOE cryptographically erase the DEK.

The evaluator shall ensure the TSS describes the process to initiate TOE firmware/software updates.

If item d) is selected in FMT_SMF.1.1: If additional management functions are claimed in the ST, the evaluator shall verify that the TSS describes those functions.

Guidance
The evaluator shall review the AGD guidance and shall determine that the instructions for changing a DEK exist. The instructions must cover all environments on which the TOE is claiming conformance, and include any preconditions that must exist in order to successfully generate or re-generate the DEK.

The evaluator shall examine the operational guidance to ensure that it describes how to initiate TOE firmware/software updates.

If Disable Key Recovery is selected: The guidance for disabling key recovery shall be described in the AGD documentation.

KMD
If item d)) is selected in FMT_SMF.1.1: If the TOE offers the functionality to import an encrypted DEK, the evaluator shall ensure the KMD describes how the TOE imports a wrapped DEK and performs the decryption of the wrapped DEK.

Tests

The evaluator shall verify that the TOE has the functionality to change and cryptographically erase the DEK (effectively removing the ability to retrieve previous user data).

The evaluator shall verify that the TOE has the functionality to initiate TOE firmware/software updates.

If item d) is selected in FMT_SMF.1.1: If additional management functions are claimed, the evaluator shall verify that the additional features function as described.

5.1.4 Protection of the TSF (FPT)

FPT_KYP_EXT.1 Protection of Key and Key Material

FPT_KYP_EXT.1.1

The TSF shall [**selection**:

- *not store keys in non-volatile memory*
- *only store keys in non-volatile memory when [**selection**: wrapped, as specified in FCS_COP.1/KeyWrap, encrypted, as specified in FCS_COP.1/KeyEnc, encrypted as specified in FCS_COP.1/KeyEncap]*
- *only store plaintext keys that meet any one of the following criteria [**selection**:*
  - *the plaintext key is not part of the key chain as specified in FCS_KYC_EXT.2*
  - *the plaintext key will no longer provide access to the encrypted data after initial provisioning*
  - *the plaintext key is a key split that is combined as specified in FCS_SMC_EXT.1, and the other half of the key split is [**selection**:*
    - *wrapped as specified in FCS_COP.1/KeyWrap*
    - *encrypted as specified in [**selection**: FCS_COP.1/KeyEnc, FCS_COP.1/KeyEncap]*
    - *derived and not stored in non-volatile memory*

    ]
  - *the non-volatile memory the key is stored on is located in an external storage device for use as an authorization factor*
  - *the plaintext key is only used to provide additional cryptographic protection to other keys, such that disclosure of the plaintext key would not compromise the security of the keys being protected*

]

].

Application Note: The plaintext key storage in non-volatile memory is allowed for several reasons. If the keys exist within protected memory that is not user accessible on the TOE or OE, the only methods that allow it to play a security relevant role for protecting the BEV or the DEK are if it is a key split or providing additional layers of wrapping or encryption on keys that have already been protected.

If the TSF implements key wrapping, key encryption, or key encapsulation to maintain protected cryptographic key storage, then FCS_COP.1/KeyWrap, FCS_COP.1/KeyEnc, or FCS_COP.1/KeyEncap must be claimed. If the TSF implements submask combining to maintain protected cryptographic key storage, then FCS_SMC_EXT.1 must be claimed. These selection must align with FCS_KYC_EXT.2.

Evaluation Activities

FPT_KYP_EXT.1
TSS
The evaluator shall examine the TSS to verify that it describes the method by which intermediate keys are generated using submask combining.

Guidance
There are no AGD evaluation activities for this SFR.

KMD
The evaluator shall examine the KMD for a description of the methods used to protect keys stored in non-volatile memory.

The evaluator shall verify the KMD to ensure it describes the storage location of all keys and the protection of all keys stored in non-volatile memory. The description of the key chain shall be reviewed to ensure the selected method is followed for the storage of wrapped or encrypted keys in non-volatile memory and plaintext keys in non-volatile memory meet one of the criteria for storage.

Tests
If "only store keys in non-volatile memory when wrapped or encrypted" is selected, the evaluator shall verify that the TOE protects key material in a way consistent with the selections and description in the TSS and KMD. This can be achieved by inspecting non-volatile locations where the TOE stores key material or by inspecting the TOE memory when loading those materials from non-volatile storage. It would be sufficient to demonstrate that the stored value is different from the plaintext value of the key as discovered in the test EA for FCS_CKM.6.

FPT_PWR_EXT.1 Power Saving States

FPT_PWR_EXT.1.1

The TSF shall define the following compliant power saving states: [**selection**: *S3, S4, G2(S5), G3, D0, D1, D2, D3, [**assignment**: other power saving states]*].

Application Note: Power saving states S3, S4, G2(S5), G3, D0, D1, D2, and D3 are defined by the Advanced Configuration and Power Interface (ACPI) standard.

Evaluation Activities

FPT_PWR_EXT.1
TSS
The evaluator shall validate the TSS contains a list of compliant power saving states.

Guidance
The evaluator shall ensure that guidance documentation contains a list of compliant power saving states. If additional power saving states are supported, then the evaluator shall validate that the guidance documentation states how the use of non-compliant power saving states can be avoided.

KMD
There is no KMD for this activity.

Tests
The evaluator shall confirm that for each listed compliant state all key/key materials are removed from

volatile memory by using the test indicated by the selection in FCS_CKM_EXT.6.

FPT_PWR_EXT.2 Timing of Power Saving States

FPT_PWR_EXT.2.1

For each compliant power saving state defined in FPT_PWR_EXT.1.1, the TSF shall enter the compliant power saving state when the following conditions occur: user-initiated request, [**selection**: *shutdown, user inactivity, request initiated by remote management system, [**assignment**: other conditions], no other conditions*].

Application Note: If volatile memory is not erased as part of an unexpected power shutdown sequence then guidance documentation must define mitigation activities (e.g. how long users should wait after an unexpected power-down before volatile memory can be considered erased).

Evaluation Activities

FPT_PWR_EXT.2

TSS

The evaluator shall validate that the TSS contains a list of conditions under which the TOE enters a compliant power saving state.

Guidance

The evaluator shall check that the guidance contains a list of conditions under which the TOE enters a Compliant power saving state. Additionally, the evaluator shall verify that the guidance documentation provides information on how long it is expected to take for the TOE to fully transition into the Compliant power saving state (e.g. how many seconds for the volatile memory to be completely erased).

KMD

There is no KMD for this activity.

Tests

The evaluator shall trigger each condition in the list of identified conditions and ensure the TOE ends up in a compliant power saving state by using the test indicated by the selection in FCS_CKM_EXT.6.

FPT_TST.1 TSF Self-Testing

FPT_TST.1.1

The TSF shall run a suite of the following self-tests [**selection**: *during initial start-up, at the conditions [before the function is first invoked]*] to demonstrate the correct operation of [**selection**: *[**assignment**: parts of TSF], the TSF*]: [**assignment**: *list of self-tests run by the TSF*].

FPT_TST.1.2

The TSF shall provide authorized users with the capability to verify the integrity of [**selection**: *[**assignment**: parts of TSF data], TSF data*].

FPT_TST.1.3

The TSF shall provide authorized users with the capability to verify the integrity of [**selection**: *[**assignment**: parts of TSF], TSF*].

Application Note: This SFR is a required dependency of FCS_RBG.1 and the cryptographic requirements that are selectable in FCS_KYC_EXT.2. It is intended to require that any DRBG implemented by the TOE undergo health testing to ensure that the random bit generation functionality has not been degraded. If the TSF supports multiple DRBGs, this SFR should be iterated to describe the self-test behavior for each.

The tests regarding cryptographic functions implemented in the TOE can be deferred, as long as the tests are performed before the function is invoked.

If any FCS_COP functions are implemented by the TOE, the TSS should describe the known answer self-tests for those functions.

Evaluation Activities

FPT_TST.1

TSS

The evaluator shall examine the TSS to ensure that it details the self-tests that are run by the TSF along with how they are run. This description should include an outline of what the tests are actually doing. The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the DRBG is operating correctly.

Note that this information may also be placed in the entropy documentation specified by Appendix D - Entropy Documentation and Assessment.

Guidance

If a self-test can be executed at the request of an authorized user, the evaluator shall verify that the operational guidance provides instructions on how to execute that self-test.

Tests

For each self-test, the evaluator shall verify that evidence is produced that the self-test is executed when specified by FPT_TST.1.1.

If a self-test can be executed at the request of an authorized user, the evaluator shall verify that following the steps documented in the operational guidance to perform the self-test will result in execution of the self-test.

FPT_TUD_EXT.1 Trusted Update

FPT_TUD_EXT.1.1

The TSF shall provide [*authorized users*] the ability to query the current version of the TOE [**selection**: *software*, *firmware*].

FPT_TUD_EXT.1.2

The TSF shall provide [*authorized users*] the ability to initiate updates to TOE [**selection**: *software*, *firmware*].

FPT_TUD_EXT.1.3

The TSF shall verify updates to the TOE [**selection**: *software*, *firmware*] using a [**selection**: *digital signature **as specified in FCS_COP.1/SigVer***, *authenticated firmware update mechanism as described in FPT_FUA_EXT.1*] by the manufacturer prior to installing those updates.

Application Note: While this component requires the TOE to implement the update functionality itself, it is acceptable to perform the cryptographic checks using functionality available in the Operational Environment.

Evaluation Activities

FPT_TUD_EXT.1
TSS
If "authenticated firmware update mechanism as described in FPT_FUA_EXT.1" is selected, then FPT_FUA_EXT.1 must be claimed.

The evaluator shall examine the TSS to ensure that it describes information stating that an authorized source signs TOE updates and will have an associated digital signature. The evaluator shall examine the TSS contains a definition of an authorized source along with a description of how the TOE uses public keys for the update verification mechanism in the Operational Environment. The evaluator ensures the TSS contains details on the protection and maintenance of the TOE update credentials.

If the Operational Environment performs the signature verification, then the evaluator shall examine the TSS to ensure it describes, for each platform identified in the ST, the interfaces used by the TOE to invoke this cryptographic functionality.

Guidance
The evaluator ensures that the operational guidance describes how the TOE obtains vendor updates to the TOE; the processing associated with verifying the digital signature of the updates (as defined in FCS_COP.1/SigVer); and the actions that take place for successful and unsuccessful cases.

KMD
There are no KMD evaluation activities for this SFR.

Tests
The evaluators shall perform the following tests (if the TOE supports multiple signatures, each using a different hash algorithm, then the evaluator performs tests for different combinations of authentic and unauthentic digital signatures and hashes, as well as for digital signature alone):

- *Test FPT_TUD_EXT.1:1: The evaluator performs the version verification activity to determine the current version of the TOE.*
- *Test FPT_TUD_EXT.1:2: The evaluator obtains a legitimate update using procedures described in the operational guidance and verifies that an update successfully installs on the TOE. The evaluator performs the version verification activity again to verify that the version correctly corresponds to that of the update.*

5.1.5 TOE Security Functional Requirements Rationale

The following rationale provides justification for each SFR for the TOE, showing that the SFRs are suitable to address the specified threats:

Table 5: SFR Rationale

Threat	Addressed by	Rationale

T.AUTHORIZATION_GUESSING	FCS_SNI_EXT.1	Mitigates this threat by requiring proper salts, which will prevent pre-computed attacks.
	FCS_VAL_EXT.1	Mitigates this threat by requiring several options for enforcing validation, such as key sanitization of the DEK or when a configurable number of failed validation attempts is reached within a 24 hour period. This mitigates brute force attacks.
T.CHOSEN_PLAINTEXT	FCS_COP.1/SKC	Mitigates this threat by ensuring the proper choice of encryption algorithm and mode.
	FCS_SNI_EXT.1	Mitigates this threat by ensuring proper handling of salts, nonces, and initialization vectors.
T.KEYING_MATERIAL_COMPROMISE	FCS_CKM.1/DEK	Mitigates this threat by ensuring that a sufficiently strong DEK is used to prevent its value from being determined.
	FCS_COP.1/SKC	Mitigates this threat by performing data encryption and decryption.
	FCS_CKM_EXT.6	Mitigates this threat by ensuring proper key material destruction.
	FCS_CKM.6/Power	Mitigates this threat by ensuring proper key material destruction for system power states.
	FCS_COP.1/Hash	Mitigates this threat by performing cryptographic hashing services.
	FCS_KYC_EXT.2	Mitigates this threat by requiring chaining of keys to protect the BEV.
	FCS_SNI_EXT.1	Mitigates this threat by requiring additional obfuscation to the protected key material by introducing IV's and salting.
	FCS_VAL_EXT.1	Mitigates this threat by defining methods for validation of the BEV and number of validation attempts.
	FMT_SMF.1	Mitigates this threat by ensuring the TSF provides the functions necessary to manage important aspects of the TOE including generating new keys.
	FPT_KYP_EXT.1	Mitigates this threat by requiring unwrapped key material is not stored in non-volatile memory.
	FPT_PWR_EXT.1	Mitigates this threat by requiring the TOE to meet a compliant power saving state that protects and/or destroys key material.
	FPT_PWR_EXT.2	Mitigates this threat by requiring the TOE to enter into a safe power saving state based on each condition.

FCS_CKM.1/AKG (selection-based)	Mitigates this threat by requiring asymmetric key generation.
FCS_CKM.6/KEK (optional)	Mitigates this threat by ensuring proper key cryptographic erase.
FCS_CKM.1/SKG (implementation-dependent)	Mitigates this threat by requiring symmetric cryptographic key generation.
FCS_CKM.6/GENHW (selection-based)	Mitigates this threat by ensuring proper key material destruction in general hardware.
FCS_CKM.6/SW (selection-based)	Mitigates this threat by ensuring proper key material destruction within the software TOE and 3rd party storage.
FCS_CKM.6/TOEHW (selection-based)	Mitigates this threat by ensuring proper key material destruction.
FCS_COP.1/KeyedHash (selection-based)	Mitigates this threat by performing cryptographic keyed-hash message authentication.
FCS_COP.1/KeyEnc (selection-based)	Mitigates this threat by performing key encryption and decryption.
FCS_COP.1/KeyWrap (selection-based)	Mitigates this threat by performing key wrapping.
FCS_CKM.5 (selection-based)	Mitigates this threat by ensuring strong key derivation methods.
FCS_RBG.1 (selection-based)	Mitigates this threat by randomizing the generated keys in order to reduce the likelihood of guessing the future keys.
FCS_RBG.2 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
FCS_RBG.3 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
FCS_RBG.4 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
FCS_RBG.5 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
FCS_SMC_EXT.1 (selection-based)	Mitigates this threat by obscuring the submasks via a XOR or hashing operation.
FPT_FLS.1 (selection-based)	Mitigates this threat by ensuring that a malfunctioning DRBG function cannot be used to generate potentially

		insecure keys.
	FPT_TST.1 (selection-based)	Mitigates this threat by verifying the cryptographic functionality through the self testing functionality.
T.KEYSPACE_EXHAUST	FCS_CKM.1/DEK	Mitigates this threat by defining a sufficiently large keyspace such that keyspace exhaust to determine the value of the DEK is computationally infeasible.
	FCS_KYC_EXT.2	Mitigates this threat by ensuring all keys protecting the BEV are of the same strength.
	FCS_CKM.1/AKG (selection-based)	Mitigates this threat by requiring asymmetric key generation.
	FCS_CKM.1/SKG (implementation-dependent)	Mitigates this threat by requiring symmetric cryptographic key generation.
	FCS_RBG.1 (selection-based)	Mitigates this threat by ensuring that keys used for data encryption are generated using a secure DRBG.
	FCS_RBG.2 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
	FCS_RBG.3 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
	FCS_RBG.4 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
	FCS_RBG.5 (selection-based)	Mitigates this threat by ensuring that the TOE's DRBG is seeded with sufficient entropy to ensure the generation of strong cryptographic keys.
	FPT_FLS.1 (selection-based)	Mitigates this threat by ensuring that a malfunctioning DRBG function cannot be used to generate potentially insecure keys.
	FPT_TST.1 (selection-based)	Mitigates this threat by verifying the cryptographic functionality through the self testing functionality.
T.KNOWN_PLAINTEXT	FCS_COP.1/SKC	Mitigates this threat by ensuring the proper choice of encryption algorithm and mode.
	FCS_SNI_EXT.1	Mitigates this threat by ensuring proper handling of salts, nonces, and initialization vectors.
T.UNAUTHORIZED_DATA_ACCESS	FCS_COP.1/SKC	Mitigates this threat by defining proper AES encryption.
	FCS_SNI_EXT.1	Mitigates this threat by ensuring proper nonces and IVs are used in the encryption of data.

	FCS_VAL_EXT.1	Mitigates this threat by verifying the correct authentication and limits attempts to decrypt the data.
	FDP_DSK_EXT.1	Mitigates this threat by ensuring the TOE performs full drive encryption, which includes all protected data.
	FMT_SMF.1	Mitigates this threat by ensuring the TSF provides the functions necessary to manage important aspects of the TOE including requests to change and erase the DEK.
	FPT_PWR_EXT.1	Mitigates this threat by defining what power states are compliant for the TOE.
	FPT_PWR_EXT.2	Mitigates this threat by defining conditions in which the TOE will enter a compliant power state. These requirements ensure the device is secure if lost in a compliant power state.
T.UNAUTHORIZED_ FIRMWARE_MODIFY	FPT_TUD_EXT.1	Mitigates this threat by providing a mechanism used to initiate updates where the authenticity and integrity of the update can be verified as part of the update process.
	FPT_FUA_EXT.1 (selection-based)	Mitigates this threat by ensuring existing firmware cannot be modified unless replaced with a valid update initiated as part of FPT_TUD_EXT.1
	FCS_COP.1/Hash	Mitigates this threat by defining the cryptographic functions that can be used to validate the authenticity and integrity of firmware updates as defined by FPT_FUA_EXT.1.
	FCS_COP.1/SigVer	Mitigates this threat by defining the cryptographic functions that can be used to validate the authenticity and integrity of firmware updates as defined by FPT_FUA_EXT.1.
T.UNAUTHORIZED_ UPDATE	FCS_COP.1/SigVer	Mitigates this threat by defining the signature function that is used to verify updates.
	FMT_SMF.1	Mitigates this threat by ensuring the TSF provides the functions necessary to manage important behavior of the TOE which includes the initiation of system firmware/software updates.
	FPT_TUD_EXT.1	Mitigates this threat by providing authorized users the ability to query the current version of the TOE software/firmware, initiate updates, and verify updates prior to installation using a manufacturer digital signature.
	FPT_FAC_EXT.1 (optional)	Mitigates this threat by providing additional security by only allowing an update to be initiated by an authorized user.
	FPT_RBP_EXT.1 (optional)	Mitigates this threat by protecting against a malicious or inadvertent downgrade of the software/firmware to

an earlier version that may have security flaws not present in the more recent version.

5.2 Security Assurance Requirements

This cPP 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. Individual evaluation activities to be performed are specified within each SFR.

Note to ST authors: There is a selection in the ASE_TSS that must be completed. One cannot simply reference the SARs in this cPP.

The general model for evaluation of TOEs against STs written to conform to this cPP is as follows: after the ST has been approved for evaluation, the ITSEF will obtain the TOE, supporting environmental IT (if required), and the administrative/user guides for the TOE. 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 Evaluation Activities contained within the SD, which are intended to be an interpretation of the other CEM assurance requirements as they apply to the specific technology instantiated in the TOE. The Evaluation Activities that are captured in the SD also provide clarification as to what the developer needs to provide to demonstrate the TOE is compliant with the cPP.

Table 6: TOE Security Assurance Requirements

Functional Class	Functional Components
Security Target (ASE)	Conformance Claims (ASE_CCL.1)
	Extended Components Definition (ASE_ECD.1)
	ST Introduction (ASE_INT.1)
	Security Objectives for the Operational Environment (ASE_OBJ.1)
	Stated Security Requirements (ASE_REQ.1)
	Security Problem Definition (ASE_SPD.1)
	TOE Summary Specification (ASE_TSS.1)
Development (ADV)	Basic Functional Specification (ADV_FSP.1)
Guidance Documents (AGD)	Operational User Guidance (AGD_OPE.1)
	Preparative Procedures (AGD_PRE.1)
Life Cycle Support (ALC)	Labeling of the TOE (ALC_CMC.1)
	TOE CM Coverage (ALC_CMS.1)
Tests (ATE)	Independent Testing – Sample (ATE_IND.1)
Vulnerability Assessment (AVA)	Vulnerability Survey (AVA_VAN.1)

5.2.1 ASE: Security Target

The ST is evaluated as per ASE activities defined in the CEM. In addition, there may be Evaluation Activities specified within the SD that call for necessary descriptions to be included in the TSS that are specific to the TOE technology type.

The SFRs in this cPP allow for conformant implementations to incorporate a wide range of acceptable key management approaches as long as basic principles are satisfied. Given the criticality of the key management scheme, this cPP requires the developer to provide a detailed description of their key management implementation. This information can be submitted as an appendix to the ST and marked proprietary, as this level of detailed information is not expected to be made publicly available. See Appendix E for details on the expectation of the developer's Key Management Description

In addition, if the TOE includes a random bit generator Appendix D provides a description of the information expected to be provided regarding the quality of the entropy.

ASE_TSS.1.1C The TOE summary specification shall describe how the TOE meets each SFR, **including a Key Management Description (Appendix E), and *[selection: Entropy Essay, list of all of 3rd party software libraries (including version numbers), 3rd party hardware components (including model/version numbers), no other cPP specified proprietary documentation]*.**

5.2.2 ADV: Development

The design information about the TOE is contained in the guidance documentation available to the end user as well as the TSS portion of the ST, and any additional information required by this cPP that is not to be made public (e.g., Entropy Essay).

ADV_FSP.1 Basic Functional Specification (ADV_FSP.1)

The functional specification describes the TOE Security Functions Interfaces (TSFIs). It is not necessary to have a formal or complete specification of these interfaces. Additionally, because TOEs conforming to this cPP may have interfaces to the Operational Environment that are not directly invoked by TOE 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 cPP, the evaluation activities for this family 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 evaluation activities specified.

The evaluation activities are associated with the applicable SFRs. 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.

Developer action elements:

ADV_FSP.1.1D

The developer shall provide a functional specification.

ADV_FSP.1.2D

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 evaluation 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.

Content and presentation elements:

ADV_FSP.1.1C

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

ADV_FSP.1.2C

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

ADV_FSP.1.3C

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

ADV_FSP.1.4C

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

Evaluator action elements:

ADV_FSP.1.1E

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

ADV_FSP.1.2E

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 evaluation 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 evaluation 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 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
- Instructions to provide a protected administrative capability.

Guidance pertaining to particular security functionality must also be provided; requirements on such guidance are contained in the evaluation activities

AGD_OPE.1 Operational User Guidance (AGD_OPE.1)

The operational user guidance does not have to be contained in a single document. Guidance to users, administrators, and integrators can be spread among documents or web pages.

The developer should review the evaluation activities 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.

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. Where appropriate, the guidance documentation is expressed in the eXtensible Configuration Checklist Description Format (XCCDF) to support security automation. Rather than repeat information here, the developer should review the evaluation 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.1.1C

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.1.2C

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

AGD_OPE.1.3C

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.

AGD_OPE.1.4C

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.1.5C

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

AGD_OPE.1.6C

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.1.7C

The operational user guidance shall be clear and reasonable.

Evaluator action elements:

AGD_OPE.1.1E

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 will be verified by the evaluation activities in Section 5.1 Security Functional Requirements and evaluation of the TOE according to the [CEM]. The following additional information is also required.

If cryptographic functions are provided by the TOE, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the TOE. 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 TOE.

The documentation must describe the process for verifying updates to the TOE by verifying a digital signature – this may be done by the TOE or the underlying platform.

The evaluator shall 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 TOE (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 digital signature. The TOE will likely contain security functionality that does not fall in the scope of evaluation under this cPP. 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)

As with the operational guidance, the developer should look to the Evaluation Activities to determine the required content with respect to preparative procedures.

Developer action elements:

AGD_PRE.1.1D

The developer shall provide the TOE, including its preparative procedures.

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

Content and presentation elements:

AGD_PRE.1.1C

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

AGD_PRE.1.2C

The preparative procedures shall describe all the steps necessary for secure installation of the TOE 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.1.1E

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

AGD_PRE.1.2E

The evaluator shall apply the preparative procedures to confirm that the TOE 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 TOE functional requirements. The evaluator shall check to ensure that the guidance provided for the TOE adequately addresses all platforms claimed for the TOE in the ST.

5.2.4 Class ALC: Life-cycle Support

At the assurance level provided for TOEs conformant to this cPP, life-cycle support is limited to end-user-visible aspects of the life-cycle, rather than an examination of the TOE 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 Labelling of the TOE (ALC_CMC.1)

This component is targeted at identifying the TOE 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. The evaluator performs the CEM work units associated with ALC_CMC.1.

Developer action elements:

ALC_CMC.1.1D

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

Content and presentation elements:

ALC_CMC.1.1C

The application shall be labeled with a unique reference.

Application Note: Unique reference information includes:

- Application Name
- Application Version
- Application Description
- Platform on which Application Runs
- Software Identification (SWID) tags, if available

Evaluator action elements:

ALC_CMC.1.1E

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

Evaluation Activities

ALC_CMC.1
The evaluator shall 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 shall check the operational guidance and TOE samples received for testing to ensure that the version number is consistent with that in the ST. If the vendor maintains a website advertising the TOE, the evaluator shall examine the information on the website to ensure that the information in the ST is sufficient to distinguish the product.

ALC_CMS.1 TOE CMS Coverage (ALC_CMS.1)

Given the scope of the TOE and its associated evaluation evidence requirements, the evaluator performs the CEM work units associated with ALC_CMS.1.

Developer action elements:

ALC_CMS.1.1D
The developer shall provide a configuration list for the TOE.

Content and presentation elements:

ALC_CMS.1.1C
The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.

ALC_CMS.1.2C
The configuration list shall uniquely identify the configuration items.

Evaluator action elements:

ALC_CMS.1.1E
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 cPP 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 TOE is specifically identified and that this identification is consistent in the ST and in the AGD guidance (as done in the evaluation 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.

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. For this cPP, 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 operational guidance (includes “evaluated configuration” instructions). The focus of the testing is to confirm that the requirements specified in Section 5 are being met. The Evaluation Activities in the SD identify the specific testing activities necessary to verify compliance with the SFRs. The evaluator produces a test report documenting the plan for and results of testing, as well as coverage arguments focused on the platform/TOE combinations that are claiming conformance to this cPP.

Developer action elements:

ATE_IND.1.1D

The developer shall provide the TOE for testing.

Application Note: The developer must provide at least one product instance of the TOE for complete testing on at least one platform regardless of equivalency. See the Equivalency Appendix for more details.

Content and presentation elements:

ATE_IND.1.1C

The TOE shall be suitable for testing.

Evaluator action elements:

ATE_IND.1.1E

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

ATE_IND.1.2E

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

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

Evaluation Activities

ATE_IND.1

The evaluator shall prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this cPP’s evaluation activities.

While it is not necessary to have one test case per test listed in an evaluation 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 effect; 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 TOE 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 cPP and used by the cryptographic protocols being evaluated (e.g., SSH). 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 current generation of this cPP, the iTC is expected to survey open sources to discover what vulnerabilities have been discovered in these types of products and provide that content into the AVA_VAN discussion. In most cases, these vulnerabilities will require sophistication beyond that of a basic attacker. This information will be used in the development of future Protection Profiles.

If the TOE is a Network Attached Storage (NAS) device, the evaluator shall verify as part of the vulnerability assessment that remote management services are either not present or can be fully disabled.

AVA_VAN.1 Vulnerability Survey (AVA_VAN.1)

Vulnerability Analysis

Sources of Vulnerability Information

CEM Work Unit AVA_VAN.1-3 is supplemented here to provide a better-defined set of flaws to investigate and procedures to follow based on this particular technology. Terminology used is based on the flaw hypothesis methodology, where the evaluation team hypothesizes flaws and then either proves or disproves those flaws (a flaw is equivalent to a "potential vulnerability" as used in the CEM). Flaws are categorized into four "types" depending on how they are formulated:

1. A list of flaw hypotheses applicable to the technology described by the cPP derived from public sources as documented in the Type 1 Hypotheses section—this fixed set has been agreed to by the iTC. Additionally, this will be supplemented with entries for a set of public sources (as indicated below) that are directly applicable to the TOE or its identified components (as defined by the process in the Type 1 Hypotheses section below); this is to ensure that the evaluators include in their assessment applicable entries that have been discovered since the cPP was published;
2. A list of flaw hypotheses contained in this document that are derived from lessons learned specific to that technology and other iTC input (that might be derived from other open sources and vulnerability databases, for example) as documented in the Type 2 Hypotheses section;
3. A list of flaw hypotheses derived from information available to the evaluators; this includes the baseline evidence provided by the vendor described in this document (documentation associated with EAs, documentation described in the Vulnerability Survey section), as well as other information (public and/or based on evaluator experience) as documented in the Type 3 Hypotheses section; and
4. A list of flaw hypotheses that are generated through the use of iTC-defined tools (e.g., nmap, protocol testers) and their application is specified in the Type 4 Hypotheses section.

Type 1 Hypotheses-Public-Vulnerability-Based

The following list of public sources of vulnerability information was selected by the iTC:

a. Search Common Vulnerabilities and Exposures: http://cve.mitre.org/cve/
b. Search the National Vulnerability Database: https://nvd.nist.gov/
c. Search US-CERT: http://www.kb.cert.org/vuls/html/search

The list of sources above was searched with the following search terms:

- General (for all)
  - Product name
  - Underlying components (e.g., OS, software libraries (crypto libraries), chipsets)
  - Drive encryption, disk encryption
  - Key destruction and sanitization
- For Software FDE (AA or EE)
  - Key caching

In order to successfully complete this activity, the evaluator will use the developer provided list of all of third party library information that is used as part of their product, along with the version and any other identifying information (this is required in the cPP as part of the ASE_TSS.1.1C requirement). This applies to hardware (including chipsets, etc.) that a vendor utilizes as part of their TOE. This TOE-unique information will be used in the search terms the evaluator uses in addition to those listed above.

The evaluator will also consider the requirements that are chosen and the appropriate guidance that is tied to each requirement.

In order to supplement this list, the evaluators shall also perform a search on the sources listed above to determine a list of potential flaw hypotheses that are more recent that the publication date of the cPP, and those that are specific to the TOE and its components as specified by the additional documentation mentioned above. Any duplicates – either in a specific entry, or in the flaw hypothesis that is generated from an entry from the same or a different source – can be noted and removed from consideration by the evaluation team.

As part of type 1 flaw hypothesis generation for the specific components of the TOE, the evaluator shall also search the component manufacturer's websites to determine if flaw hypotheses can be generated on this basis (for instance, if security patches have been released for the version of the component being evaluated, the subject of those patches may form the basis for a flaw hypothesis).

Type 2 Hypotheses-iTC-Sourced

There are no type 2 hypotheses.

Type 3 Hypotheses-Evaluation-Team-Generated

The iTC has leveraged the expertise of the developers and the evaluation labs to diligently develop the appropriate search terms and vulnerability databases. They have also thoughtfully considered the iTC-sourced hypotheses the evaluators should use based upon the applicable use case and the threats to be mitigated by the SFRs. Therefore, it is the intent of the iTC, for the evaluation to focus all effort on the Type 1 and Type 2 Hypotheses and has decided that Type 3 Hypotheses are not necessary.

However, if the evaluators discover a Type 3 potential flaw that they believe should be considered, they should work with their Certification Body to determine the feasibility of pursuing the hypothesis. The Certification Body may determine whether the potential flaw hypotheses is worth submitting to the iTC for consideration as Type 2 hypotheses in future drafts of the cPP/SD.

Type 4 Hypotheses-Evaluation-Team-Generated

The iTC has called out several tools that should be used during the Type 2 hypotheses process. Therefore, the use of any tools is covered within the Type 2 construct and the iTC does not see any additional tools that are necessary. The use case for Version 2 of this cPP is rather straightforward – the device is found in a powered down state and has not been subjected to revisit/evil maid attacks. Since that is the use case, the iTC has also assumed there is a trusted channel between the AA and EE. Since the use case is so narrow, and is not a typical model for penetration or fuzzing testing, the normal types of testing do not apply. Therefore, the relevant types of tools are referenced in Type 2.

Process for Evaluator Vulnerability Analysis

As flaw hypotheses are generated from the activities described above, the evaluation team will disposition them; that is, attempt to prove, disprove, or determine the non-applicability of the hypotheses. This process is as follows. The evaluator will refine each flaw hypothesis for the TOE and attempt to disprove it using the information provided by the developer or through penetration testing. During this process, the evaluator is free to interact directly with the developer to determine if the flaw exists, including requests to the developer for additional evidence (e.g., detailed design information, consultation with engineering staff); however, the CB should be included in these discussions. Should the developer object to the information being requested as being not compatible with the overall level of the evaluation activity/cPP and cannot provide evidence otherwise that the flaw is disproved, the evaluator prepares an appropriate set of materials as follows:

- The source documents used in formulating the hypothesis, and why it represents a potential compromise against a specific TOE function;
- An argument why the flaw hypothesis could not be proven or disproved by the evidence provided so far; and
- The type of information required to investigate the flaw hypothesis further.

The Certification Body (CB) will then either approve or disapprove the request for additional information. If approved, the developer provides the requested evidence to disprove the flaw hypothesis (or, of course, acknowledge the flaw).

For each hypothesis, the evaluator will note whether the flaw hypothesis has been successfully disproved, successfully proven to have identified a flaw, or requires further investigation. It is important to have the results documented as outlined in the Reporting section below. If the evaluator finds a flaw, the evaluator must report these flaws to the developer. All reported flaws must be addressed as follows:

If the developer confirms that the flaw exists and that it is exploitable at Basic Attack Potential, then a change is made by the developer, and the resulting resolution is agreed by the evaluator and noted as part of the evaluation report.

If the developer, the evaluator, and the CB agree that the flaw is exploitable only above Basic Attack Potential and does not require resolution for any other reason, then no change is made and the flaw is noted as a residual vulnerability in the CB-internal report (ETR).

If the developer and evaluator agree that the flaw is exploitable only above Basic Attack Potential, but it is deemed critical to fix because of technology-specific or cPP-specific aspects such as typical use cases or operational environments, then a change is made by the developer, and the resulting resolution is agreed by the evaluator and noted as part of the evaluation report.

Disagreements between evaluator and vendor regarding questions of the existence of a flaw, its attack potential, or whether it should be deemed critical to fix are resolved by the CB.

Any testing performed by the evaluator shall be documented in the test report as outlined in the Reporting section below.

As indicated in the Reporting section, the public statement with respect to vulnerability analysis that is

performed on TOEs conformant to the cPP is constrained to coverage of flaws associated with Types 1 and 2 (defined in the Sources of Vulnerability Information section) flaw hypotheses only. The fact that the iTC generates these candidate hypotheses indicates these must be addressed.

Reporting

The evaluators shall produce two reports on the testing effort; one that is public-facing (that is, included in the non-proprietary evaluation report, which is a subset of the Evaluation Technical Report (ETR)), and the complete ETR that is delivered to the overseeing CB.

The public-facing report contains:

- The flaw identifiers returned when the procedures for searching public sources were followed according to instructions in the Type 1 Hypotheses section;
- A statement that the evaluators have examined the Type 1 flaw hypotheses specified in this document in the Type 1 Hypotheses section (i.e. the flaws listed in the previous bullet) and the Type 2 flaw hypotheses specified in the the Type 2 Hypotheses section

No other information is provided in the public-facing report.

The internal CB report contains, in addition to the information in the public-facing report:

- a list of all of the flaw hypotheses generated (cf. AVA_VAN.1-4);
- the evaluator penetration testing effort, outlining the testing approach, configuration, depth and results (cf. AVA_VAN.1-9);
- all documentation used to generate the flaw hypotheses (in identifying the documentation used in coming up with the flaw hypotheses, the evaluation team must characterize the documentation so that a reader can determine whether it is strictly required in this document, and the nature of the documentation (design information, developer engineering notebooks, etc.));
- the evaluator shall report all exploitable vulnerabilities and residual vulnerabilities, detailing for each:
    a. its source (e.g. CEM activity being undertaken when it was conceived, known to the evaluator, read in a publication)
    b. the SFRs not met;
    c. a description;
    d. whether it is exploitable in its operational environment or not (i.e. exploitable or residual).
    e. the amount of time, level of expertise, level of knowledge of the TOE, level of opportunity and the equipment required to perform the identified vulnerabilities (cf. AVA_VAN.1-11);
    f. how each flaw hypothesis was resolved (this includes whether the original flaw hypothesis was confirmed or disproved, and any analysis relating to whether a residual vulnerability is exploitable by an attacker with Basic Attack Potential) (cf. AVA_VAN.1-10); and
    g. in the case that actual testing was performed in the investigation (either as part of flaw hypothesis generation using tools specified by the iTC in the Type 4 Hypotheses section, or in proving or disproving a particular flaw) the steps followed in setting up the TOE (and any required test equipment); executing the test; post-test procedures; and the actual results (to a level of detail that allow repetition of the test, including the following:
        - identification of the potential vulnerability the TOE is being tested for;
        - instructions to connect and setup all required test equipment as required to conduct the penetration test;
        - instructions to establish all penetration test prerequisite initial conditions;
        - instructions to stimulate the TSF;
        - instructions for observing the behaviour of the TSF;
        - descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results;
        - instructions to conclude the test and establish the necessary post-test state for the TOE. (cf. AVA_VAN.1-6, AVA_VAN.1-8).

Developer action elements:

AVA_VAN.1.1D

The developer shall provide the TOE for testing.

Content and presentation elements:

AVA_VAN.1.1C

The application shall be suitable for testing.

Application Note: Suitability for testing means not being obfuscated or packaged in such a way as to disrupt either static or dynamic analysis by the evaluator.

Evaluator action elements:

AVA_VAN.1.1E

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

AVA_VAN.1.2E

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

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.1.3E

The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing Basic attack potential.

Evaluation Activities

AVA_VAN.1

The evaluator shall generate a report to document their findings with respect to this requirement. This report could physically be part of the overall test report mentioned in ATE_IND, or a separate document. The evaluator performs a search of public information to find vulnerabilities that have been found in similar applications with a particular focus on network protocols the application uses and document formats it parses.

The evaluator documents the sources consulted and the vulnerabilities found in the report.

For each vulnerability found, the evaluator either provides a rationale with respect to its non-applicability, or the evaluator formulates a test (using the guidelines provided in ATE_IND) to confirm the vulnerability, if suitable. Suitability is determined by assessing the attack vector needed to take advantage of the vulnerability. If exploiting the vulnerability requires expert skills and an electron microscope, for instance, then a test would not be suitable and an appropriate justification would be formulated.

Appendix A - Optional Requirements

As indicated in the introduction to this PP, the baseline requirements (those that must be performed by the TOE) are contained in the body of this PP. This appendix contains three other types of optional requirements:

The first type, defined in Appendix A.1 Strictly Optional Requirements, are strictly optional requirements. If the TOE meets any of these requirements the vendor is encouraged to claim the associated SFRs in the ST, but doing so is not required in order to conform to this PP.

The second type, defined in Appendix A.2 Objective Requirements, are objective requirements. These describe security functionality that is not yet widely available in commercial technology. Objective requirements are not currently mandated by this PP, but will be mandated in the future. Adoption by vendors is encouraged, but claiming these SFRs is not required in order to conform to this PP.

The third type, defined in Appendix A.3 Implementation-dependent Requirements, are Implementation-dependent requirements. If the TOE implements the product features associated with the listed SFRs, either the SFRs must be claimed or the product features must be disabled in the evaluated configuration.

A.1 Strictly Optional Requirements

A.1.1 Class ALC: Life-cycle Support

ALC_FLR.1 Basic Flaw Remediation (ALC_FLR.1)

This SAR is optional and may be claimed at the ST-Author's discretion.

Developer action elements:

ALC_FLR.1.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

Content and presentation elements:

ALC_FLR.1.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

ALC_FLR.1.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

ALC_FLR.1.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

ALC_FLR.1.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

Evaluator action elements:

ALC_FLR.1.1E

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

Evaluation Activities

ALC_FLR.1
The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.

ALC_FLR.2 Flaw Reporting Procedures (ALC_FLR.2)

This SAR is optional and may be claimed at the ST-Author's discretion.

Developer action elements:

ALC_FLR.2.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

ALC_FLR.2.2D

The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws.

ALC_FLR.2.3D

The developer shall provide flaw remediation guidance addressed to TOE users.

Content and presentation elements:

ALC_FLR.2.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

ALC_FLR.2.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

ALC_FLR.2.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

ALC_FLR.2.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

ALC_FLR.2.5C

The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE.

ALC_FLR.2.6C

The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users.

ALC_FLR.2.7C

The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

ALC_FLR.2.8C

The flaw remediation guidance shall describe a means by which TOE users report to the developer any suspected security flaws in the TOE.

Evaluator action elements:

ALC_FLR.2.1E

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

Evaluation Activities

ALC_FLR.2
The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.
The evaluator shall inspect the guidance document and verify it describes how to access the flaw remediation guidance.

ALC_FLR.3 Systematic Flaw Remediation (ALC_FLR.3)

This SAR is optional and may be claimed at the ST-Author's discretion.

Developer action elements:

ALC_FLR.3.1D

The developer shall document and provide flaw remediation procedures addressed to TOE developers.

ALC_FLR.3.2D

The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws.

ALC_FLR.3.3D

The developer shall provide flaw remediation guidance addressed to TOE users.

Content and presentation elements:

ALC_FLR.3.1C

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.

ALC_FLR.3.2C

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

ALC_FLR.3.3C

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

ALC_FLR.3.4C

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.

ALC_FLR.3.5C

The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE.

ALC_FLR.3.6C

The flaw remediation procedures shall include a procedure requiring timely response and the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.

ALC_FLR.3.7C

The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users.

ALC_FLR.3.8C

The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

ALC_FLR.3.9C

The flaw remediation guidance shall describe a means by which TOE users report to the developer any suspected security flaws in the TOE.

ALC_FLR.3.10C

The flaw remediation guidance shall describe a means by which TOE users may register with the developer, to be eligible to receive security flaw reports and corrections.

ALC_FLR.3.11C

The flaw remediation guidance shall identify the specific points of contact for all reports and enquiries about security issues involving the TOE.

Evaluator action elements:

ALC_FLR.3.1E

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

Evaluation Activities

ALC_FLR.3

The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.

The evaluator shall inspect the guidance document and verify it describes how to access the flaw remediation guidance.

A.1.2 Protection of the TSF (FPT)

FPT_FAC_EXT.1 Firmware / Software Access Control

FPT_FAC_EXT.1.1

The TSF shall require [**selection**: *a password, a unique value printed on the device, a authorized user action*] before the [**selection**: *software, firmware*] update proceeds.

Application Note: Before an update takes place, the TSF owner will authorize the update by providing either a unique value (for example, a serial number) that is printed on the drive, a password (which should be administratively configurable as defined in FMT_SMF.1), or perform a specific action as an authorized user. If applicable, it is assumed that physical presence to the drive is limited to authorized personnel. If the correct value is not provided, the update will not take place. The values are intended to be unique per drive so they cannot be easily guessed.

Evaluation Activities

FPT_FAC_EXT.1
TSS
The evaluator shall examine the TSS to ensure that it describes information stating how the access control process takes place along with a description of the values or actions that are used.

Guidance
The evaluator ensures that the Operational Guidance describes how the user will be expected to interact with the authorization process.

KMD
There is no KMD for this activity.

Tests
The evaluator shall perform the following test.

- *Test FPT_FAC_EXT.1:1: The evaluator shall try installing a upgrade and verify that a prompt is required and the appropriate value or action is necessary for the update to continue.*

FPT_RBP_EXT.1 Rollback Protection

FPT_RBP_EXT.1.1

The TSF shall verify that the new update is not downgrading to a lower security version number by [**assignment**: *method of verifying the security version number is the same as or higher than the currently installed version*].

FPT_RBP_EXT.1.2

The TSF shall generate and return an error code if the attempted update package is detected to be an invalid version.

Application Note: This requirement prevents an unauthorized rollback of the update to an earlier authentic version. This mitigates against unknowing installation of an earlier authentic version that may have a security weakness. It is expected that vendors will increase security version numbers with each new update package.

For FPT_RBP_EXT.1.1 the purpose is to verify that the new package has a security version number equal to or larger than the security version number of currently

installed package.

The administrator guidance would include instructions for the administrator to configure the rollback protection mechanism, if appropriate.

The security version may be different from other parts that undergo update by the TSF.

Evaluation Activities

FPT_RBP_EXT.1
TSS
The evaluator shall examine the TSS to ensure that it describes at a high level the process for verifying that security version checking is performed before an upgrade is installed. The evaluator shall verify that a high level description of the types of error codes are provided and when an error would be triggered.

Guidance
The evaluator ensures that a description is provided on how the user should interpret the error codes.

KMD
There is no KMD for this activity.

Tests
The evaluator shall perform the following test:

- *Test FPT_RBP_EXT.1:1: The evaluator shall try installing a lower security version number upgrade (either by just modifying the version number or by using an upgrade provided by the vendor) and will verify that the lower version cannot be installed and an error is presented to the user.*

A.2 Objective Requirements

This PP does not define any Objective requirements.

A.3 Implementation-dependent Requirements

A.3.1 Cryptographic Support (FCS)

FCS_CKM.1/SKG Cryptographic Key Generation (Symmetric Keys)

This component must be included in the ST if the TOE implements any of the following features:

- ***Symmetric Key Generation Support***

FCS_CKM.1.1/SKG

The TSF shall generate **symmetric** cryptographic keys **using a Random Bit Generator as specified in FCS_RBG.1** and specified cryptographic key sizes *256 bit* that meet the following: [*no standard*].

Application Note: Symmetric keys may be used to generate keys along the key chain. This applies to any instance in where the TSF DRBG is referenced for key generation where the TSF generates or re-generates key encryption or key wrapping keys as part of deriving a key (as in FCS_KYC_EXT.2)

Evaluation Activities

FCS_CKM.1/SKG

TSS

The evaluator shall review the TSS to determine that a symmetric key is supported by the product, that the TSS includes a description of the protection provided by the product for this key. The evaluator shall ensure that the TSS identifies the key sizes supported by the TOE.

Guidance

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key sizes for all uses specified by the AGD documentation and defined in this cPP.

KMD

If the TOE uses a symmetric key as part of the key chain, the KMD should detail how the symmetric key is used as part of the key chain.

Tests

There are no test evaluation activities for this SFR.

Appendix B - Selection-based Requirements

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

B.1 Cryptographic Support (FCS)

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

The inclusion of this selection-based component depends upon selection in:

- ***FCS_COP.1.1/KeyEncap***

FCS_CKM.1.1/AKG

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

- *CNSA 2.0 Compliant Algorithms: [**selection**:*
  - ***Module-Lattice-Based Key-Encapsulation Mechanism** using the parameter set ML-KEM-1024*
  - ***Module-Lattice-Based Digital Signature Algorithm** using the parameter set ML-DSA-87*

  ]
- *CNSA 1.0 Compliant Algorithms: [**selection**:*
  - ***[RSA schemes]** using cryptographic key sizes of [**selection**: 3072, 4096, 6144, 8192]*
  - ***[ECC schemes]** using ["NIST curves" P-384 and [**selection**: P-521, no other curves]]*

  ]

] that meet the following: [**selection**: *FIPS PUB 203 [ML-KEM], FIPS PUB 204 [ML-DSA], FIPS PUB 186-5 Appendix A.1 [RSA], FIPS PUB 186-5 Appendix A.2 [ECC]*].

Application Note: Asymmetric keys may be used to "wrap" a key or submask. This SFR should be included by the ST author when making the appropriate selection in FCS_COP.1/KeyEncap.

Note that ML-DSA and ML-KEM are not usable in any functions at the time of initial publication, they are added to this requirement in support of future protocol updates. As support is expanded for CNSA 2.0, CNSA 1.0 will be removed as an selection in a future update.

Evaluation Activities

FCS_CKM.1/AKG
TSS

The evaluator shall ensure that the TSS identifies the key sizes / parameter sets supported by the TOE. If the ST specifies more than one scheme, the evaluator shall examine the TSS to verify that it identifies the usage for each scheme.

Guidance

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key generation schemes and key sizes for all uses specified by the AGD documentation and defined in this cPP.

KMD

If the TOE uses an asymmetric key as part of the key chain, the KMD should detail how the asymmetric key is used as part of the key chain.

Tests

*If the ST selects "**implement functionality**," then the following test activities shall be carried out.*

Evaluation Activity Note: The following tests may require the developer to provide access to a developer environment that provides the evaluator with tools that are not typically available to end-users of the application

Key Generation for FIPS PUB 186-5 RSA Schemes

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. Key Pair generation specifies 5 ways (or methods) to generate the primes p and q. These include:

- *Random Primes:*
  - *Provable primes*
  - *Probable primes*
- *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. This includes the random seed(s), the public exponent of the RSA key, and the desired key length. 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 from a known good implementation.

If possible, the Random Probable primes method should also be verified against a known good implementation as described above. Otherwise, the evaluator shall have the TSF generate 10 keys pairs for each supported key length nlen and verify:

- $n = p \cdot q$,
- *p and q are probably prime according to Miller-Rabin tests,*
- $GCD(p-1, e) = 1$,
- $GCD(q-1, e) = 1$,
- $2^{16} \leq e \leq 2^{256}$ *and e is an odd integer,*
- $|p-q| > 2^{nlen/2 - 100}$,
- $p \geq 2^{nlen/2 -1/2}$,
- $q \geq 2^{nlen/2 -1/2}$,
- $2^{(nlen/2)} < d < LCM(p-1, q-1)$,

- *e·d = 1 mod LCM(p-1, q-1).*

Key Generation for Elliptic Curve Cryptography (ECC)

FIPS 186-5 ECC Key Generation Test*- For each supported NIST curve, i.e., P-384 and P-521, the evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator shall submit the generated key pairs to the public key verification (PKV) function of a known good implementation.*

FIPS 186-5 Public Key Verification (PKV) Test*- For each supported NIST curve, i.e., P-384 and P-521, the evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify five 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.*

Key Generation for ML-DSA

The evaluator shall 10x input to the internal KeyGen function a 32-byte random seed. Verify the returned public-private key pair is correct using a known good implementation. Here internal KeyGen refers to the TOE's implementation of the function ML-DSA.KeyGen_internal(-) as described in FIPS.204.

Key Generation for ML-KEM

The evaluator shall 10x input to the internal KeyGen function a pair of 32-byte random string. Verify the returned encapsulation and decapsulation key pair is correct using a known good implementation. Here internal KeyGen refers to the TOE's implementation of the function ML-KEM.KeyGen_internal(-,-) as described in FIPS.203.

FCS_CKM.6/GENHW Cryptographic Key Destruction (General Hardware)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM_EXT.6.1***

FCS_CKM.6.1/GENHW

The TSF shall destroy [**assignment**: *list of cryptographic keys (including keying material)*] when [*no longer needed*].

FCS_CKM.6.2/GENHW

The TSF shall destroy cryptographic keys and keying material specified in FCS_CKM.6.1/GENHW in accordance with a specified cryptographic key destruction method [**selection**:

- *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,*

- *destruction of reference to the key directly followed by a request for garbage collection*

]

- *For non-volatile memory, the destruction shall be executed by a [**selection**:*
  - *single*
  - *[**assignment**: ST author defined multi-pass]*

  *] overwrite consisting of [**selection**:*
  - *a pseudo-random pattern using the TSF's RBG,*
  - *zeroes,*
  - *ones,*
  - *a new value of a key of the same size,*
  - *[**assignment**: some value that does not contain any CSP]*
  - *block erase*

  ]

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

Application Note: This SFR must be included in the ST if selected in FCS_CKM_EXT.6.

In the first selection, the ST Author is presented options for destroying disused cryptographic keys based on whether they are in volatile memory or non-volatile storage within the TOE. The selection of block erase for non-volatile storage applies only to flash memory. A block erase does not require a read-verify, since the reference to the memory location is erased as well as the data itself.

Within the selections is the option to overwrite the memory location with a new value of a key. The intent is that a new value of a key (as specified in another SFR within the cPP) can be used to "replace" an existing key.

Several selections allow assignment of a 'value that does not contain any CSP'. This means that the TOE uses some other specified data not drawn from an RBG meeting FCS_RBG.1 requirements, and not being any of the particular values listed as other selection options. The point of the phrase 'does not contain any CSP' is to ensure that the overwritten data is carefully selected, and not taken from a general 'pool' that might contain current or residual data that itself requires confidentiality protection.

Key destruction does not apply to the public component of asymmetric key pairs.

Evaluation Activities

FCS_CKM.6/GENHW
TSS
(Key Management Description may be used if necessary details describe proprietary information)

The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

The evaluator shall check to ensure the TSS lists each type of key that is stored, and identifies the memory type (volatile or non-volatile) where key material is stored.

The TSS identifies and describes the interfaces that is used to service commands to read/write memory. The evaluator examines the interface description for each different media type to ensure that the interface supports the selections made by the ST Author.

If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator shall verify that the pattern does not contain any CSPs.

The evaluator shall check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement.

Guidance

There are a variety of concerns that may prevent or delay key 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 any other relevant Required Supplementary Information. The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer.

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, it is assumed the drive supports the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. It is assumed the operating system and file system of the OE support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion.

It is assumed that if a RAID array is being used, only set-ups that support TRIM are utilized. It is assumed if the drive is connected via PCI-Express, the operating system supports TRIM over that channel. It is assumed the drive is healthy and contains minimal corrupted data and will be end of life before a significant amount of damage to drive health occurs, it is assumed there is a risk small amounts of potentially recoverable data may remain in damaged areas of the drive.

Finally, it is assumed the keys are not stored using a method that would be inaccessible to TRIM, such as being contained in a file less than 982 bytes which would be completely contained in the master file table.

Tests

For these tests the evaluator shall utilize appropriate development environment (e.g. a Virtual Machine) and development tools (debuggers, simulators, etc.) to test that keys are erased, including all copies of the key that may have been created internally by the TOE during normal cryptographic processing with that key.

- *Test FCS_CKM.6/GENHW:1: Applied to each key 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). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator shall:*
    1. *Record the value of the key in the TOE subject to erasure.*
    2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
    3. *Cause the TOE to erase the key.*
    4. *Cause the TOE to stop the execution but not exit.*

5. *Cause the TOE to dump the entire memory of the TOE into a binary file.*
6. *Search the content of the binary file created in Step #5 for instances of the known key value from Step #1.*
7. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece.*

Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

Step 7 ensures that partial key fragments do not remain in memory. If a fragment is found, there is a miniscule 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 found the test fails.

- *Test FCS_CKM.6/GENHW:2: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:*
  1. *Record the value of the key in the TOE subject to erasure.*
  2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to erase the key.*
  4. *Search the non-volatile memory the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.*
  5. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece. If a fragment is found then the test is repeated (as described for test 1 above), and if a fragment is found in the repeated test then the test fails.*
- *Test FCS_CKM.6/GENHW:3: Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:*
  1. *Record the storage location of the key in the TOE subject to erasure.*
  2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to erase the key.*
  4. *Search the storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is used.*

The test succeeds if the correct pattern is used to overwrite the key in the memory location. If the pattern is not found, the test fails.

FCS_CKM.6/KEK Cryptographic Key Destruction (Key Cryptographic Erase)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM_EXT.6.1***

FCS_CKM.6.1/KEK

The TSF shall destroy [**assignment**: *list of cryptographic keys (including keying material)*] when [*no longer needed*].

FCS_CKM.6.2/KEK

The TSF shall destroy cryptographic keys and keying material specified by FCS_CKM.6.1/KEK in accordance with a specified cryptographic key destruction method [*by using another method in FCS_CKM_EXT.6.1 to destroy all encryption keys encrypting the key intended for destruction*] that meets the following: [*no standard*].

Application Note: A key can be considered destroyed by destroying the key that protects the key. If a key is wrapped or encrypted it is not necessary to “overwrite” that key, overwriting the key that is used to wrap or encrypt the key used to encrypt/decrypt

data, using the appropriate method for the memory type involved, will suffice. For example, if a product uses a Key Encryption Key (KEK) to encrypt a Data Encryption Key (DEK), destroying the KEK using one of the methods in FCS_CKM_EXT.6.1 is sufficient, since the DEK would no longer be usable (of course, presumes the DEK is still encrypted).

Evaluation Activities

FCS_CKM.6/KEK
TSS
These is no TSS for this activity.

Guidance
There is no AGD for this activity.

KMD
The evaluator shall examine the TOE's keychain in the TSS/KMD and identify each instance a key is destroyed by this method. In each instance the evaluator shall verify all keys capable of decrypting the target key are destroyed in accordance with a specified key destruction method.

Tests
There is no test for this activity.

FCS_CKM.6/SW Cryptographic Key Destruction (Software TOE, 3rd Party Storage)

The inclusion of this selection-based component depends upon selection in:
- ***FCS_CKM_EXT.6.1***

FCS_CKM.6.1/SW

The TSF shall destroy [**assignment**: *list of cryptographic keys (including keying material)*] when [*no longer needed*].

FCS_CKM.6.2/SW

The TSF shall destroy cryptographic keys and keying material specified in FCS_CKM.6.1/SW in accordance with a specified cryptographic key destruction method [**selection**:

- *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,*
  - *destruction of reference to the key directly followed by a request for garbage collection*

]

- *For non-volatile storage that consists of the invocation of an interface provided by the underlying platform that [**selection**:*
  - *logically addresses the storage location of the key and performs a [**selection**: single, [**assignment**: ST author defined multi-pass]] overwrite consisting of [**selection**:*
    - *a pseudo-random pattern using the TSF's RBG,*
    - *zeroes,*
    - *ones,*
    - *a new value of a key of the same size,*
    - *[**assignment**: some value that does not contain any CSP]*

    ]
  - *instructs the underlying platform to destroy the abstraction that represents the key*

]

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

Application Note: This SFR must be included in the ST if selected in FCS_CKM_EXT.6.

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 visible. For instance, in a given implementation the application may have access to the file system details and may be able to logically address specific memory locations. In another implementation the application may simply have a handle to a resource and can only ask the platform to delete the resource. The level of detail to which the TOE has access will be reflected in the TSS section of the ST.

Several selections allow assignment of a 'value that does not contain any CSP'. This means that the TOE uses some other specified data not drawn from an RBG meeting FCS_RBG.1 requirements, and not being any of the particular values listed as other selection options. The point of the phrase 'does not contain any CSP' is to ensure that the overwritten data is carefully selected, and not taken from a general 'pool' that might contain current or residual data that itself requires confidentiality protection.

Key destruction does not apply to the public component of asymmetric key pairs.

Evaluation Activities

FCS_CKM.6/SW
TSS
(Key Management Description may be used if necessary details describe proprietary information)

The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g., by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

The evaluator shall check to ensure the TSS lists each type of key that is stored in non-volatile memory, and identifies how the TOE interacts with the underlying platform to manage keys (e.g., store, retrieve, destroy). The description includes details on the method of how the TOE interacts with the platform, including an identification and description of the interfaces it uses to manage keys (e.g., file system

APIs, platform key store APIs).

The evaluator examines the interface description for each different media type to ensure that the interface supports the selections and description in the TSS.

The evaluator shall check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement. If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator shall verify that the pattern does not contain any CSPs.

Guidance
There are a variety of concerns that may prevent or delay key 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 any other relevant Required Supplementary Information. The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer.

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, it is assumed the drive supports the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. It is assumed the operating system and file system of the OE support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion.

It is assumed that if a RAID array is being used, only set-ups that support TRIM are utilized. It is assumed if the drive is connected via PCI-Express, the operating system supports TRIM over that channel. It is assumed the drive is healthy and contains minimal corrupted data and will be end of life before a significant amount of damage to drive health occurs, it is assumed there is a risk small amounts of potentially recoverable data may remain in damaged areas of the drive.

Finally, it is assumed the keys are not stored using a method that would be inaccessible to TRIM, such as being contained in a file less than 982 bytes which would be completely contained in the master file table.

Tests

- *Test FCS_CKM.6/SW:1: Applied to each key 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). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator shall:*
    1. *Record the value of the key in the TOE subject to erasure.*
    2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
    3. *Cause the TOE to erase the key.*
    4. *Cause the TOE to stop the execution but not exit.*
    5. *Cause the TOE to dump the entire memory of the TOE into a binary file.*
    6. *Search the content of the binary file created in Step #5 for instances of the known key value from Step #1.*
    7. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece.*

Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

Step 7 ensures that partial key fragments do not remain in memory. If a fragment is found, there is a miniscule 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 found the test fails.

The following tests apply only to selection a), since the TOE in this instance has more visibility into what is happening within the underlying platform (e.g., a logical view of the media). In selection b), the TOE has no visibility into the inner workings and completely relies on the underlying platform, so there is no reason to test the TOE beyond test 1.

For selection a), the following tests are used to determine the TOE is able to request the platform to overwrite the key with a TOE supplied pattern.

- *Test FCS_CKM.6/SW:2: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media (e.g., MBR file system):*
  1. *Record the value of the key in the TOE subject to erasure.*
  2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to erase the key.*
  4. *Search the logical view that the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.*
  5. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece. If a fragment is found then the test is repeated (as described for Use Case 1 test 1 above), and if a fragment is found in the repeated test then the test fails.*
- *Test FCS_CKM.6/SW:3: Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media:*
  1. *Record the logical storage location of the key in the TOE subject to erasure.*
  2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to erase the key.*
  4. *Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is used.*

  The test succeeds if the correct pattern is used to overwrite the key in the memory location. If the pattern is not found, the test fails.

FCS_CKM.6/TOEHW Cryptographic Key Destruction (TOE-Controlled Hardware)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM_EXT.6.1***

FCS_CKM.6.1/TOEHW

The TSF shall destroy [**assignment**: *list of cryptographic keys (including keying material)*] when [*no longer needed*].

FCS_CKM.6.2/TOEHW

The TSF shall destroy cryptographic keys and keying material specified in FCS_CKM.6.1/TOEHW in accordance with a specified cryptographic key destruction method [**selection**:

- *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,*
  - *destruction of reference to the key directly followed by a request for garbage collection*

  ]
- *For non-volatile [**selection**:*
  - *that employs a wear-leveling algorithm, the destruction shall be executed by a [**selection**: single overwrite consisting of zeroes, single overwrite consisting of ones, overwrite with a new value of a key of the same size, single overwrite consisting of [**assignment**: some value that does not contain any CSP], a pseudo-random pattern using the TSF's RBG, 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 zeros followed by a read-verify]*
    - *[**selection**: single, [**assignment**: ST author defined multi-pass] overwrite consisting of ones followed by a read-verify]*
    - *[**selection**: single, [**assignment**: ST author defined multi-pass]overwrite consisting of [**assignment**: some value that does not contain any CSP] followed by a read-verify]block erase*

    ]

  ] and 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.

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

Application Note: This SFR must be included in the ST if selected in FCS_CKM_EXT.6.

In the first selection, the ST Author is presented options for destroying a key based on the memory or storage technology where keys are stored within the TOE.

If non-volatile memory is used to store keys, the ST Author selects whether the memory storage algorithm uses wear-leveling or not. Storage technologies or memory types that use wear-leveling are not required to perform a read verify. The selection for destruction includes block erase as an option, and this option applies only to flash memory. A block erase does not require a read verify, since the mappings of logical addresses to the erased memory locations are erased as well as the data itself.

Within the selections is the option to overwrite a disused key with a new value of a key. The intent is that a new value of a key (as specified in another SFR within the cPP) can be used to "replace" an existing key.

If a selection for read verify is chosen, it should generate an audit record upon failures.

Several selections allow assignment of a ‘value that does not contain any CSP’. This means that the TOE uses some other specified data not drawn from an RBG meeting FCS_RBG.1 requirements, and not being any of the particular values listed as other selection options. The point of the phrase ‘does not contain any CSP’ is to ensure that the overwritten data is carefully selected, and not taken from a general ‘pool’ that might contain current or residual data that itself requires confidentiality protection.

Key destruction does not apply to the public component of asymmetric key pairs.

Evaluation Activities

FCS_CKM.6/TOEHW
TSS
(Key Management Description may be used if necessary details describe proprietary information)

The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g., by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

The evaluator shall examine the TSS to ensure it describes the method that is used by the memory controller to write and read memory from each type of memory listed. The purpose here is to provide a description of how the memory controller works so one can determine exactly how keys are written to memory. The description would include how the data is written to and read from memory (e.g., block level, cell level), mechanisms for copies of the key that could potentially exist (e.g., a copy with parity bits, a copy without parity bits, any mechanisms that are used for redundancy).

The evaluator shall examine the TSS to ensure it describes the destruction procedure for each key that has been identified. If different types of memory are used to store the keys, the evaluator shall check to ensure that the TSS identifies the destruction procedure for each memory type where keys are stored (e.g., key X stored in flash memory is destroyed by overwriting once with zeros, key X’ stored in EEPROM is destroyed by a overwrite consisting of a pseudo random pattern – the EEPROM used in the TOE uses a wear-leveling scheme as described).

If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator shall verify that the pattern does not contain any CSPs.

The evaluator shall check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement.

Upon completion of the TSS examination, the evaluator understands how all the keys (and potential copies) are destroyed.

Guidance
There are a variety of concerns that may prevent or delay key 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 any other relevant Required Supplementary Information. The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer.

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies

of the key that are logically inaccessible but persist physically. In this case, it is assumed the drive supports the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. It is assumed the operating system and file system of the OE support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion.

It is assumed that if a RAID array is being used, only set-ups that support TRIM are utilized. It is assumed if the drive is connected via PCI-Express, the operating system supports TRIM over that channel. It is assumed the drive is healthy and contains minimal corrupted data and will be end of life before a significant amount of damage to drive health occurs, it is assumed there is a risk small amounts of potentially recoverable data may remain in damaged areas of the drive.

Finally, it is assumed the keys are not stored using a method that would be inaccessible to TRIM, such as being contained in a file less than 982 bytes which would be completely contained in the master file table.

For destruction on wear-leveled memory, if a time period is required before is processed destruction the ST author shall provide an estimated range.

Tests
For these tests the evaluator shall use appropriate development environment (e.g. a Virtual Machine) and development tools (debuggers, simulators, etc.) to test that keys are erased, including all copies of the key that may have been created internally by the TOE during normal cryptographic processing with that key.

For destruction on wear-leveled memory, if a time period is required before is evaluator shall wait that amount of time after erasing the key in tests 2 and 3.

- *Test FCS_CKM.6/TOEHW:1: Applied to each key 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). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator shall:*
    1. *Record the value of the key in the TOE subject to erasure.*
    2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
    3. *Cause the TOE to erase the key.*
    4. *Cause the TOE to stop the execution but not exit.*
    5. *Cause the TOE to dump the entire memory of the TOE into a binary file.*
    6. *Search the content of the binary file created in Step #5 for instances of the known key value from Step #1.*
    7. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece.*

  Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

  Step 7 ensures that partial key fragments do not remain in memory. If a fragment is found, there is a miniscule 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 found the test fails.
- *Test FCS_CKM.6/TOEHW:2: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:*

1. *Record the value of the key in the TOE subject to erasure.*
2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
3. *Cause the TOE to erase the key.*
4. *Search the non-volatile memory the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.*
5. *Break the key value from Step #1 into 3 or 4 similar sized pieces and perform a search using each piece. If a fragment is found then the test is repeated (as described for Use test 1 above), and if a fragment is found in the repeated test then the test fails.*

- *Test FCS_CKM.6/TOEHW:3: Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:*
  1. *Record the logical storage location of the key in the TOE subject to erasure.*
  2. *Cause the TOE to perform a normal cryptographic processing with the key from Step #1.*
  3. *Cause the TOE to erase the key.*
  4. *Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is used.*

  The test succeeds if the correct pattern is used to overwrite the key in the memory location. If the pattern is not found, the test fails.

FCS_COP.1/KeyEncap Cryptographic Operation - Key Encapsulation

The inclusion of this selection-based component depends upon selection in:

- ***FCS_KYC_EXT.2.2*,**
- ***FPT_KYP_EXT.1.1***

FCS_COP.1.1/KeyEncap

The TSF shall perform [key encapsulation] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 7 provides the allowed choices for completion of the selection operations of FCS_COP.1/KeyEncap.

Table 7: Allowed choices for FCS_COP.1/KeyEncap

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
KAS1	RSASVE	[**selection**: *3072, 4096, 6144, 8192*] bits	NIST SP 800-56B Revision 2
KTS-OAEP	RSA-OAEP	[**selection**: *3072, 4096, 6144, 8192*] bits	NIST SP 800-56B Revision 2
ML-KEM	ML-KEM	Parameter set = ML-KEM-1024	NIST FIPS 203

Application Note: This requirement is used in the body of the ST if the ST author chooses to use key transport in the key chaining approach that is specified in FCS_KYC_EXT.2.

NIST SP 800-57 Part 1 Revision 5 Section 5.6.2 specifies that the size of key used to

protect the key being transported should be at least the security strength of the key it is protecting.

If this SFR is claimed, then FCS_CKM.1/AKG must also be claimed.

KAS1 and KTS-OAEP with the selectable parameters are CNSA 1.0 compliant. ML-KEM-1024 is CNSA 2.0 compliant.

Evaluation Activities

FCS_COP.1/KeyEncap

TSS

The evaluator shall ensure that the TSS documents that the selection of the key size is sufficient for the security strength of the key encapsulated.

The evaluator shall examine the TSS to verify that any one-time values such as nonces or masks are constructed and used in accordance with the relevant standards.

Guidance

There are no AGD evaluation activities for this SFR.

KMD

There are no KMD evaluation activities for this SFR.

Tests

The following tests may 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.

KAS1 [RSASVE single-party]

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
KAS1	RSASVE	[**selection:** 3072, 4096, 6144, 8192] bits	NIST SP 800-56B Revision 2 (Section 7.2.1)

To test the TOE's implementation of the of KAS1 RSASVE Single-Party Key Encapsulation, the evaluator shall perform the Algorithm Functional Test and Validation Test using the following input parameters:

- *RSA Private key format [Basic with fixed public exponent, Prime Factor with fixed public exponent, Chinese Remainder Theorem with fixed public exponent, Basic with random public exponent, Prime Factor with random public exponent, Chinese Remainder Theorem with random public exponent]*
- *Modulo value [3072, 4096, 6144, 8192]*
- *Role [initiator, responder]*
- *Key confirmation supported [yes, no]*

The evaluator shall generate a test group (i.e. set of tests) for each parameter value of the above parameter type with the largest number of supported values. For example, if the TOE supports all six key formats, then the evaluator shall generate six test groups. Each of the above supported parameter values must be included in at least one test group.

Regardless of how many parameter values are supported, there must be at least two test groups.

Half of the test groups are designated as Algorithm Functional Tests (AFT) and the remainder are designated as Validation Tests (VAT). If there is an odd number of groups, then the extra group is designated randomly as either AFT or VAT.

If there are only two test groups, in addition to the above, one shall act as an initiator, and the other as a responder, if supported.

Algorithm Functional Test

For each test group designated as AFT, the evaluator shall generate 10 test cases using random data (except for a fixed public exponent, if supported). The resulting shared secrets shall be compared with those generated by a known-good implementation using the same inputs.

Validation Test

For each test group designated as VAT, the evaluator shall generate 25 test cases are using random data (except for a fixed public exponent, if supported). Of the 25 test cases:

- *Two test cases must have a shared secret with a leading nibble of 0s,*
- *Two test cases have modified derived key material,*
- *Two test cases have modified tags, if key confirmation is supported,*
- *Two test cases have modified MACs, if key confirmation is supported, and*
- *The remaining test cases are not modified.*

To determine correctness, the evaluator shall confirm that the resulting 25 shared secrets correspond as expected for both the modified and unmodified values.

KTS-OAEP [RSA-OAEP]

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
KTS-OAEP	RSA-OAEP	[**selection:** 3072, 4096, 6144, 8192] bits	NIST SP 800-56B Revision 2 (Sections 6.3 & 9)

To test the TOE's implementation of the of KTS-OAEP, the evaluator shall perform the Algorithm Functional Test and Validation Test using the following input parameters:

- *Roles [initiator, receiver]*
- *Private Key format [Basic with fixed public exponent, Prime Factor with fixed public exponent, Chinese Remainder Theorem with fixed public exponent, Basic with random public exponent, Prime Factor with random public exponent, Chinese Remainder Theorem with random public exponent]*
- *Supported modulus values [3072, 4096, 6144, 8192]*
- *Key confirmation supported [yes, no]*

The evaluator shall generate a test group (i.e. set of tests) for each parameter value of the above parameter type with the largest number of supported values. For example, if the TOE supports all six key formats, then the evaluator shall generate six test groups. Each of the above supported parameter values must be included in at least one test group.

Regardless of how many parameter values are supported, there must be at least two test groups.

Half of the test groups are designated as Algorithm Functional Tests (AFT) and the remainder are designated as Validation Tests (VAT). If there is an odd number of groups, then the extra group is designated randomly as either AFT or VAT.

If there are only two test groups, in addition to the above, one shall act as an initiator, and the other as a responder, if supported.

Algorithm Functional Test

For each test group designated as AFT, the evaluator shall generate 10 test cases using random data (except for a fixed public exponent, if supported). The resulting shared secrets shall be compared with those generated by a known-good implementation using the same inputs.

Validation Test

For each test group designated as VAT, the evaluator shall generate 25 test cases are using random data (except for a fixed public exponent, if supported). Of the 25 test cases:

- *Two test cases must have a shared secret with a leading nibble of 0s,*
- *Two test cases have modified derived key material,*
- *Two test cases have modified tags, if key confirmation is supported,*
- *Two test cases have modified MACs, if key confirmation is supported, and*
- *The remaining test cases are not modified.*

To determine correctness, the evaluator shall confirm that the resulting 25 shared secrets correspond as expected for both the modified and unmodified values.

ML-KEM Key Encapsulation

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
ML-KEM	ML-KEM	Parameter set = ML-KEM-1024	NIST FIPS PUB 203

To test the TOE's implementation of ML-KEM key encapsulation/decapsulation, the evaluator shall perform the Encapsulation Test and the Decapsulation Test using the following input parameters:

- *Encapsulation Parameters:*
  - *Parameter set [ML-KEM-1024]*
  - *Previously generated encapsulation key (ek)*
  - *Random value (m) [32 bytes]*
- *Decapsulation Parameters:*
  - *Parameter set [ML-KEM-1024]*
  - *Previously generated decapsulation key (dk)*
  - *Previously generated ciphertext (c) [32 bytes]*

Encapsulation Test

For each supported parameter set the evaluator shall generate 25 test cases consisting of an encapsulation key ek and random value m. For each test case the valuator shall require the implementation under test to generate the corresponding shared secret k and ciphertext c. To determine correctness, the evaluator shall compare the resulting values with those generated using a known-good implementation using the same inputs.

Encapsulation Key Check (if supported)

The evaluator shall generate 10 encapsulation keys such that:

- *Five of the encapsulation keys are valid, and*
- *Five of the encapsulation keys are modified such that a value in the noisy linear system is encoded into the key as a value greater than Q.*

The evaluator shall invoke the TOE's Encapsulation Key Check functionality to determine the validity of the 10 keys. The unmodified keys should be determined valid, and the modified keys should be determined invalid.

Decapsulation Key Check (if supported)

The evaluator shall generate 10 decapsulation keys such that:

- *Five of the decapsulation keys are valid, and*
- *Five of the decapsulation keys are modified such that the concatenated values ek||H(ek) will no longer match by modifying H(ek) to be a different value.*

The evaluator shall invoke the TOE's Decapsulation Key Check functionality to determine the validity of the 10 keys. The unmodified keys should be determined valid, and the modified keys should be determined invalid.

Decapsulation Test

For each supported parameter set the evaluator shall use a single previously generated decapsulation key dk and generate 10 test cases consisting of valid and invalid ciphertexts c. For each test case the evaluator shall require the implementation under test to generate the corresponding shared secret k whether or not the ciphertext is valid. To determine correctness, the evaluator shall compare the resulting values with those generated using a known-good implementation using the same inputs.

FCS_COP.1/KeyedHash Cryptographic Operation (Keyed Hash Algorithm)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM.5.1*,**
- ***FCS_RBG.1.1*,**
- ***FCS_VAL_EXT.1.1***

FCS_COP.1.1/KeyedHash

The TSF shall perform [keyed hash message authentication] in accordance with a specified cryptographic algorithm [**selection**: *Keyed Hash Algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 8 provides the allowed choices for completion of the selection operations of FCS_COP.1/KeyedHash.

Table 8: Allowed choices for FCS_COP.1/KeyedHash

Keyed Hash Algorithm	Cryptographic key sizes	List of standards
HMAC-SHA-256	256 bits	[**selection**: *ISO/IEC 9797-2:2021 (Section 7 "MAC Algorithm 2"), FIPS PUB 198-1*]

HMAC-SHA-384	[**selection**: *384 (ISO, FIPS), 256 (FIPS)*] bits	[**selection**: *ISO/IEC 9797-2:2021 (Section 7 "MAC Algorithm 2"), FIPS PUB 198-1*]
HMAC-SHA-512	[**selection**: *512 (ISO, FIPS), 384 (FIPS), 256 (FIPS)*] bits	[**selection**: *ISO/IEC 9797-2:2021 (Section 7 "MAC Algorithm 2"), FIPS PUB 198-1*]

Application Note: The HMAC minimum key sizes in the table are specified in ISO/IEC 9797-2:2021, which requires that the minimum key size be equal to the digest size. The FIPS standard specifies no minimum or maximum key sizes, so if FIPS PUB 198-1 is selected, larger or smaller key sizes may be used. This is indicated by the parenthesized annotations in the Cryptographic Key Sizes column.

In accordance with CNSA 1.0 and 2.0, HMAC-SHA-256 may be used only as a PRF, including as part of a key derivation function or RBG.

Evaluation Activities

FCS_COP.1/KeyedHash

TSS

The evaluator shall examine the TSS to ensure that the size of the key is sufficient for the desired security strength of the output.

The evaluator shall examine the TSS to verify that if HMAC-SHA-256 is selected, that it is being used only as a PRF in a key derivation function or RBG.

Guidance

There is no AGD for this activity.

KMD

There is no KMD for this activity.

Tests

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.

HMAC

Keyed Hash Algorithm	Cryptographic Key Sizes	List of Standards
HMAC-SHA-256	256 bits	[**selection:** ISO/IEC 9797-2:2021 (Section 7 "MAC Algorithm 2"), FIPS PUB 198-1]
HMAC-SHA-384	[**selection:** (ISO, FIPS) 384, (FIPS) 256] bits	[**selection:** ISO/IEC 9797-2:2021 (Section 7 "MAC Algorithm 2"), FIPS PUB 198-1]

HMAC-SHA-512	[**selection:** (ISO, FIPS) 512, (FIPS) 384, 256] bits	[**selection:** ISO/IEC 9797-2:2021 (Section 7 “MAC Algorithm 2”), FIPS PUB 198-1]

To test the TOE's ability to generate keyed hashes using HMAC the evaluator shall perform the Algorithm Functional Test for each combination of claimed HMAC algorithm the following parameters:

- *Hash function [SHA-256, SHA-384, SHA-512]*
- *Key length [8-65536] bits by 8s*
- *MAC length [32-[digest size of hash function (256, 384, 512)]] bits*

Algorithm Functional Test

For each supported Hash function the evaluator shall generate 150 test cases using random input messages of 128 bits, random supported key lengths, random keys, and random supported MAC lengths such that across the 150 test cases:

- *The key length includes the minimum, the maximum, a key length equal to the block size, and key lengths that are both larger and smaller than the block size.*
- *The MAC size includes the minimum, the maximum, and two other random values.*

The evaluator shall compare the output against results generated by a known-good implementation with the same input.

FCS_COP.1/KeyEnc Cryptographic Operation (Key Encryption)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_KYC_EXT.2.2*,**
- ***FPT_KYP_EXT.1.1***

FCS_COP.1.1/KeyEnc

The TSF shall perform [symmetric-key key encryption/decryption] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 9 provides the allowed choices for completion of the selection operations of FCS_COP.1/KeyEnc.

Table 9: Allowed choices for FCS_COP.1/KeyEnc

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
AES-CBC	AES in CBC mode with non-repeating and unpredictable IVs	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 10116:2017 (Clause 7), NIST*

			SP 800-38A] [CBC]
AES-GCM	AES in GCM mode with non-repeating IVs using [**selection**: *deterministic*, *RBG-based*], IV construction; the tag must be of length [**selection**: *96*, *104*, *112*, *120*, *128*] bits.	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2)*, *FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 19772:2020 (Clause 10)*, *NIST SP 800-38D*] [GCM]
AES-XTS	AES in XTS mode with unique tweak values that are consecutive non-negative integers starting at an arbitrary non-negative integer	512 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2)*, *FIPS PUB 197*] [AES] [**selection**: *IEEE Std. 1619-2018*, *NIST SP 800-38E*] [XTS]

Application Note: This SFR is required when the TSF performs key encryption as part of maintaining and deriving a key chain.

Evaluation Activities

FCS_COP.1/KeyEnc

TSS

The evaluator shall examine the TSS to ensure that it describes the construction of any IVs, tweak values, and counters in conformance with the relevant specifications.

If a GCM mode algorithm is selected, then the evaluator shall examine the TOE summary specification to confirm 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. The evaluator shall also confirm that for each invocation of GCM, the length of the plaintext is at most (2^{32})-2 blocks.

If AES-XTS is claimed then the evaluator shall examine the TSS to verify that the TOE creates full-length keys by methods that ensure that the two key halves are distinct. The evaluator shall confirm the TSS describes the block of data containing the key and that is full block of data in alignment with selected the AES standard.

Guidance

If multiple key encryption modes are supported, the evaluator examines 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 vendor's KMD to verify that it includes a description of how key encryption will be used as part of the key chain.

Tests
The AES test should be followed in FCS_COP.1/SKC Cryptographic Operation - Symmetric Key Cryptography.

FCS_COP.1/KeyWrap Cryptographic Operation - Key Wrapping

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM.1.1/DEK,***
- ***FCS_KYC_EXT.2.2,***
- ***FCS_VAL_EXT.1.1,***
- ***FPT_KYP_EXT.1.1***

FCS_COP.1.1/KeyWrap

The TSF shall perform [key wrapping] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 10 provides the allowed choices for completion of the selection operations of FCS_COP.1/KeyWrap.

Table 10: Allowed choices for FCS_COP.1/KeyWrap

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
AES-KW	AES in KW mode	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 19772:2020 (clause 6), NIST SP 800-38F (Section 6.2)*] [KW mode]
AES-KWP	AES in KWP mode	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] NIST SP 800-38F (Section 6.3) [KWP mode]
AES-CCM	AES in CCM mode with unpredictable, non-repeating	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause*

	nonce, minimum size of 64 bits		*5.2), FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 19772:2020 (Clause 7), NIST SP 800-38C*] [CCM]
AES-GCM	AES in GCM mode with non-repeating IVs using [**selection**: *deterministic*, *RBG-based*], IV construction; the tag must be of length [**selection**: *96*, *104*, *112*, *120*, *128*] bits.	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES] [**selection**: *ISO/IEC 19772:2020 (Clause 10), NIST SP 800-38D*] [GCM]

Application Note: This SFR is required when the TSF performs key wrapping as part of maintaining and deriving a key chain (FCS_KYC_EXT.2) or when the TSF performs validation of a submask, intermediate key, or BEV using a key wrap operation (FCS_VAL_EXT.1).

NIST 800-57p1rev5 sec. 5.6.2 specifies that the size of key used to protect the key being transported should be at least the security strength of the key it is protecting.

Evaluation Activities

FCS_COP.1/KeyWrap

TSS

The evaluator shall ensure that the TSS documents that the selection of the key size is sufficient for the security strength of the key wrapped.

The evaluator shall examine the TSS to ensure that it describes the construction of any IVs, nonces, and MACs in conformance with the relevant specifications.

If a CCM mode algorithm is selected, then the evaluator shall examine the TOE summary specification to confirm that it describes how the nonce is generated and that the same nonce is never reused to encrypt different plaintext pairs under the same key.

If a GCM mode algorithm is selected, then the evaluator shall examine the TOE summary specification to confirm 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. The evaluator shall also confirm that for each invocation of GCM, the length of the plaintext is at most (2^{32})-2 blocks.

Guidance

There is no AGD for this activity.

KMD

The evaluator shall review the KMD to ensure that all keys are wrapped using the approved method

and a description of when the key wrapping occurs.

Tests
For tests of AES-GCM, see testing for FCS_COP.1/SKC.

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.

AES-KW

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-KW	AES in KW mode	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] [**selection:** ISO/IEC 19772:2020 (clause 6), NIST SP 800-38F (Section 6.2)] [KW mode]

To test the TOE's ability to wrap keys using AES in Key Wrap mode the evaluator shall perform the Algorithm Functional Tests using the following input parameters:

- *Key size [256] bits*
- *Keyword cipher type [cipher, inverse]*
- *Payload sizes [128-4096] bits by 64s*

Algorithm Functional Test

The evaluator shall generate 100 encryption test cases using random data for each combination of claimed key size, keyword cipher type, and six supported payload sizes such that the payload sizes include the minimum, the maximum, two that are divisible by 128, and two that are not divisible by 128.

The results shall be compared with those generated by a known-good implementation using the same inputs.

The evaluator shall generate 100 decryption test cases using the same parameters as above, but with 20 of each 100 test cases having modified ciphertext to produce an incorrect result. To determine correctness, the evaluator shall confirm that the results correspond as expected for both the modified and unmodified values.

AES-KWP

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-KWP	AES in KWP mode	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] NIST SP 800-38F (Section 6.3) [KWP mode]

To test the TOE's ability to wrap keys using AES in Key Wrap with Padding mode with padding the evaluator shall perform the Algorithm Functional Tests using the following input parameters:

- *Key size [256] bits*
- *Keyword cipher type [cipher, inverse]*
- *Payload sizes [8-4096] bits by 8s*

Algorithm Functional Test

The evaluator shall generate 100 encryption test cases using random data for each combination of claimed key size, keyword cipher type, and six supported payload sizes such that the payload sizes include the minimum, the maximum, two that are divisible by 128, and two that are not divisible by 128.

The results shall be compared with those generated by a known-good implementation using the same inputs.

The evaluator shall generate 100 decryption test cases using the same parameters as above, but with 20 of each 100 test cases having modified ciphertext to produce an incorrect result. To determine correctness, the evaluator shall confirm that the results correspond as expected for both the modified and unmodified values.

AES-CCM

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-CCM	AES in CCM mode with non-repeating nonce, minimum size of 64 bits	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] [**selection:** ISO/IEC 19772:2020 (Clause 7), NIST SP 800-38C] [CCM]

To test the TOE's implementation of AES-CCM authenticated encryption functionality the evaluator shall perform the Algorithm Functional Tests described below using the following input parameters:

- *Key Size [256] bits*
- *Associated data size [0-65536] bits in increments of 8*
- *Payload size [0-256] bits in increments of 8*
- *IV/Nonce size [64-104] bits in increments of 8*
- *Tag size [32-128] bits in increments of 16*

Algorithm Functional Tests

Unless otherwise specified, the following tests should use random data, a tag size of 128 bits, IV/Nonce size of 104 bits, payload size of 256 bits, and associated data size of 256 bits. If any of these values are not supported, any supported value may be used. The evaluator shall compare the output from each test case against results generated by a known-good implementation with the same input parameters.

Variable Associated Data Test

For each claimed key size, and for each supported associated data size from 0 through 256 bits in increments of 8 bits, the TOE must be tested by encrypting 10 test cases using all random data. In addition, for each key size, the TOE must be tested by encrypting 10 cases with associated data lengths of 65536 bits, if supported.

Variable Payload Test

For each claimed key size, and for each supported payload size from 0 through 256 bits in increments of 8 bits, the TOE must be tested by encrypting 10 test cases using all random data.

Variable Nonce Test

For each claimed key size, and for each supported IV/Nonce size from 64 through 104 bits in increments of 8 bits, the TOE must be tested by encrypting 10 test cases using all random data.

Variable Tag Test

For each claimed key size, and for each supported tag size from 32 through 128 bits in increments of 16 bits, the TOE must be tested by encrypting 10 test cases using all random data.

Decryption Verification Test

For each claimed key size, for each supported associated data size from 0 through 256 bits in increments of 8 bits, for each supported payload size from 0 through 256 bits in increments of 8 bits, for each supported IV/Nonce size from 64 through 104 bits in increments of 8 bits, and for each supported tag size from 32 through 128 bits in increments of 16 bits, the TOE must be tested by decrypting 10 test cases using all random data.

FCS_CKM.5 Cryptographic Key Derivation

The inclusion of this selection-based component depends upon selection in:

- ***FCS_KYC_EXT.2.2***

FCS_CKM.5.1

The TSF shall derive cryptographic keys [**selection**: *Key type*] from [**selection**: *Input parameters*] in accordance with a specified key derivation algorithm [**selection**: *Key derivation algorithm*] and specified cryptographic key sizes [**selection**: *Key sizes*] that meet the following: [**selection**: *List of standards*]

Table 11 provides the allowed choices for completion of the selection operations of FCS_CKM.5.

Table 11: Allowed choices for FCS_CKM.5

Key type	Input parameters	Key derivation algorithm	Key sizes	List of standards
KDF-CTR	[**selection**: *Direct Generation from a Random Bit Generator as specified in FCS_RBG.1, Concatenated keys*]	KPF2 - KDF in Counter Mode using [**selection**: *AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-384,*	[**selection**: *256, 384, 512*] bits	[**selection**: *ISO/IEC 11770-6:2016 (Subclause 7.3.2) [KPF2], NIST SP 800-108 Revision 1 Update 1 (Section 4.1)*

		HMAC-SHA-512] as the PRF		*[KDF in Counter Mode]*]
KDF-FB	[**selection**: *Direct Generation from a Random Bit Generator as specified in FCS_RBG.1, Concatenated keys*]	KPF3 - KDF in Feedback Mode using [**selection**: *AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512*] as the PRF	[**selection**: *256, 384, 512*] bits	[**selection**: *ISO/IEC 11770-6:2016 (Subclause 7.3.3) [KPF3], NIST SP 800-108 Revision 1 Update 1 (Section 4.2) [KDF in Feedback Mode]*]
KDF-DPI	[**selection**: *Direct Generation from a Random Bit Generator as specified in FCS_RBG.1, Concatenated keys*]	KDF in Double Pipeline Iteration Mode using [**selection**: *AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512*] as the PRF	[**selection**: *256, 384, 512*]bits	[**selection**: *ISO/IEC 11770-6:2016 (Subclause 7.3.4) [KPF4], NIST SP 800-108 Revision 1 Update 1 (Section 4.3) [KDF in Double-Pipeline Iteration Mode]*]
KDF-HASH	Shared secret	Hash function [**selection**: *SHA-384, SHA-512*]	[**selection**: *256, 384, 512*] bits	NIST SP 800-56C Revision 2 (Section 4.1, Option 1) [One-Step Key Derivation]
KDF-MAC-1S	Shared secret, salt, IV, output length, fixed information	Keyed hash [**selection**: *HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512*]	[**selection**: *256, 384, 512*] bits	NIST SP 800-56C Revision 2 (Section 4.1, Options 2, 3) [One-Step Key Derivation]
KDF-MAC-2S	Shared secret, salt, IV, output length, fixed information, and [**selection**: *auxiliary shared secret, no other parameters*]	MAC Step [**selection**: *HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512*] as randomness extraction and;	[**selection**: *256, 384, 512*] bits	NIST SP 800-56C Revision 2 (Section 5) [Two-Step Key Derivation]

		KDF Step [**selection**: *KDF-CTR*, *KDF-FB*, *KDF-DPI*].		
PBKDF2	P/password/input/BEV, S/salt, IV, kLEN/output length	PRF[**selection**: *HMAC-SHA-256*, *HMAC-SHA-384*, *HMAC-SHA-512*]	[**selection**: *256*, *384*, *512*] bits	NIST Special Publication 800-132 Recommendation for Password-Based Key Derivation

Application Note: If KDF-CTR, KDF-FB, or KDF-DPI is claimed, then either FCS_COP.1/CMAC or FCS_COP.1/KeyedHash must also be claimed, depending on the selection made for PRF.

If KDF-Hash is claimed, then FCS_COP.1/Hash must also be claimed.

If KDF-MAC-1S is claimed, then FCS_COP.1/KeyedHash must also be claimed.

If KDF-MAC-2S is claimed, then both FCS_COP.1/KeyedHash and FCS_COP.1/CMAC must also be claimed.

In KDF-MAC-2S, CMAC has been removed as a selection for the MAC step because it requires selection of 128 bits for the output key size, which is not supported in CNSA 1.0. If HMAC is selected in the MAC step, then the same HMAC is used as the KDF.

The security strengths of the Pseudo-Random functions for the key derivation methods must be sufficient for the security strength of the keys derived through those methods. Since CNSA 1.0 permits keys no smaller than 256 bits, no 128- or 192-bit PRFs are permitted. If PBKDF2 is selected a salt must be generated per FCS_SNI_EXT.1.

Evaluation Activities

FCS_CKM.5

TSS

The evaluator shall verify the TSS includes a description of the key derivation function and shall verify the key derivation function uses an approved derivation mode and key expansion algorithm according to list of standards.

Guidance

The evaluator shall verify that the Guidance instructs the administrator how to set any configurable parameters, such as context strings, salts, and IVs.

The evaluator shall verify that the Guidance instructs the administrator how to configure the TOE to choose specific PRFs, modes, and parameters.

KMD

The evaluator shall verify that the KMD describes and documents:

- *that the security strengths of the Pseudo-Random functions for the key derivation methods are sufficient for the security strength of the keys derived through those methods.*
- *that the security strengths of the input parameters are sufficient for the security strength of the keys derived through these methods.*

- *that, if concatenated keys or intermediary keys are input parameters, the KMD describes the sources of the keys, and the order in which they are concatenated, along with any other values that are concatenated with them. This may occur in instances when input keying material for the KDF comes from two independent sources, for example, a client and a server.*
- *that, for KDF-MAC-1S, KDF-MAC-2S, and KDF-KMAC, the KMD documents that for each invocation of a KDF that reuses the same input shared secret or key, each invocation must use a distinct context string, IV, or salt. The KMD must also describe the composition and sizes of these input parameters. The evaluator must ensure that the context string, IV, and salt are generated in conformance with the relevant standards.*
- *that, if the TOE uses the derived key in a key chain/hierarchy, that KMD describes how the key is used as part of the key chain/hierarchy.*

Tests
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.

KDF in Counter Mode

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards
KDF-CTR	[**selection:** Direct Generation from a Random Bit Generator as specified in FCS_RBG.1, Concatenated keys]	KPF2 - KDF in Counter Mode using [**selection:** AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512] as the PRF	[**selection:** 256, 384, 512] bits	[**selection:** ISO/IEC 11770-6:2016 (Subclause 7.3.2) [KPF2], NIST SP 800-108 Revision 1 Update 1 (Section 4.1) [KDF in Counter Mode]]

To test the TOE's ability to derive cryptographic keys using KDF in Counter Mode/KDF2, the evaluator shall perform the Counter KDF Algorithm Functional Test using the following input parameters:

- *Pseudo Random Function (PRF) [AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-512]*
- *Derived key length [256, 512] bits*
- *Location of the counter [after fixed data, before fixed data, middle fixed data]*
- *Counter length [8, 16, 24, 32] bits*

Counter KDF Algorithm Functional Test
For each supported combination of the above input parameters the evaluator shall require the implementation under test to derive two keys using random data. The evaluator shall compare the resulting keys with keys generated using a known-good implementation using the same input parameters.

KDF in Feedback Mode

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards
KDF-FB	[**selection:** Direct Generation from a	KPF3 - KDF in Feedback Mode using	[**selection:** 256, 384,	[**selection:** ISO/IEC 11770-6:2016

	Random Bit Generator as specified in FCS_RBG.1, Concatenated keys]	[**selection:** AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512] as the PRF	512] bits	(Subclause 7.3.3) [KPF3], NIST SP 800-108 Revision 1 Update 1 (Section 4.2) [KDF in Feedback Mode]]

To test the TOE's ability to derive cryptographic keys using KDF in Feedback Mode/KDF3, the evaluator shall perform the Feedback KDF Algorithm Functional Test using the following input parameters:

- *Pseudo Random Function (PRF) [AES-256-CMAC, HMAC-SHA-256, HMAC-SHA-512]*
- *Derived key length [256, 512] bits*
- *Location of the counter [none, after fixed data, before fixed data, before iterator]*
- *Counter length [0, 8, 16, 24, 32] bits*

Feedback KDF Algorithm Functional Test
For each supported combination of the above input parameters the evaluator shall require the implementation under test to derive two keys using random data. The evaluator shall compare the resulting keys with keys generated using a known-good implementation using the same input parameters.

KDF in Double-Pipeline Iteration Mode

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards
KDF-DPI	[**selection:** Direct Generation from a Random Bit Generator as specified in FCS_RBG.1, Concatenated keys]	KPF4 - KDF in Double-Pipeline Iteration Mode using [**selection:** HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512] as the PRF	[**selection:** 256, 384, 512] bits	[**selection:** ISO/IEC 11770-6:2016 (Subclause 7.3.4) [KPF4], NIST SP 800-108 Revision 1 Update 1 (Section 4.3) [KDF in Double-Pipeline Iteration Mode]]

To test the TOE's ability to derive cryptographic keys using KDF in Double Pipeline Iteration Mode/KDF4, the evaluator shall perform the Double Pipeline Iteration KDF Algorithm Functional Test using the following input parameters:

- *Pseudo Random Function (PRF) [HMAC-SHA-256, HMAC-SHA-512]*
- *Derived key length [256, 512] bits*
- *Location of the counter [none, after fixed data, before fixed data, before iterator]*
- *Counter length [0, 8, 16, 24, 32] bits*

Double Pipeline Iteration KDF Algorithm Functional Test
For each supported combination of the above input parameters the evaluator shall require the implementation under test to derive two keys using random data. The evaluator shall compare the resulting keys with keys generated using a known-good implementation using the same input parameters.

KDF by Hashing a Shared Secret

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards

KDF-HASH	Shared secret	Hash function [**selection:** SHA-384, SHA-512]	[**selection:** 256, 384, 512] bits	NIST SP 800-56C Revision 2 (Section 4.1, Option 1) [One-Step Key Derivation]

To test the TOE's ability to derive cryptographic keys by hashing a shared secret (a.k.a. One-Step HASH-based Key Derivation), the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *Auxiliary Function [SHA-384, SHA-512]*
- *Derived key length [256, 384, 512] bits*

Algorithm Functional Test

For each supported fixed information pattern and combination of the above input parameters the evaluator shall require the implementation under test to derive 15 keys using random data for a shared secret that is the same size as the derived key. The evaluator shall compare the resulting keys with keys derived using a known-good implementation using the same fixed information patterns and input parameters.

One-Step MAC-based KDF

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards
KDF-MAC-1S	Shared secret, salt, output length, fixed information	Keyed Hash function [**selection:** HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512]	[**selection:** 256, 384, 512] bits	NIST SP 800-56C Revision 2 (Section 4.1, Options 2, 3) [One-Step Key Derivation]

To test the TOE's ability to derive cryptographic keys using One-Step MAC-based Key Derivation, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *Auxiliary Function [HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512]*
- *Salt [0s, random]*
- *Derived key length [256, 384, 512] bits*
- *Fixed information pattern*

Algorithm Functional Test

For each supported fixed information pattern and combination of the above input parameters the evaluator shall require the implementation under test to derive 15 keys using random data for a shared secret. The evaluator shall compare the resulting keys with keys derived using a known-good implementation using the same fixed information patterns and input parameters.

Two-Step MAC-based KDF

Key Type	Input Parameters	Key Derivation Algorithm	Key Sizes	List of Standards

KDF-MAC-2S	Shared secret, salt, IV, output length, fixed information, and [**selection:** auxiliary shared secret, no other parameters]	MAC Step [**selection:** HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512] as randomness extraction and; KDF Step [**selection:** KDF-CTR, KDF-FB, KDF-DPI]	[**selection:** 256, 384, 512] bits	NIST SP 800-56C Revision 2 (Section 5) [Two-Step Key Derivation]

To test the TOE's ability to derive cryptographic keys using Two-Step MAC-based Key Derivation, the evaluator shall perform the Algorithm Functional Test using the following input parameters:

- *MAC mode [HMAC-SHA-256; HMAC-SHA-384, HMAC-SHA-512]*
- *KDF Mode [Counter, feedback, Double Pipeline Iteration]*
- *Salt [0s, random]*
- *Length of shared secret [224-65535]*
- *Length of Auxiliary Shared Secret [0, 112-65535]*
- *Derived key length [256, 384, 512] bits*
- *Fixed information pattern*
- *Counter location [none, before fixed data, after fixed data, before iterator]*
- *Counter length [0, 8, 16, 24, 32]*

Algorithm Functional Test

The evaluator shall define a test group for each supported combination of KDF mode, MAC mode, fixed information pattern, derived key length, counter location, counter length, salt method, and five random pairs of shared secrets & auxiliary secrets (if supported) such that collectively the minimum length, maximum length and three random lengths of each are included in each test group. For each test group, the evaluator shall require the implementation under test to derive 25 keys using random data for a shared secret, either a random salt or a salt of all 0s, and, if supported, an auxiliary shared secret consisting of random data. The evaluator shall compare the resulting keys with keys derived using a known-good implementation using the same input parameters.

FCS_COP.1/CMAC Cryptographic Operation - CMAC

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM.5.1***

FCS_COP.1.1/CMAC

The TSF shall perform [CMAC] in accordance with a specified cryptographic algorithm [**selection**: *Cryptographic algorithm*] and cryptographic key sizes [**selection**: *Cryptographic key sizes*] that meet the following: [**selection**: *List of standards*]

Table 12 provides the allowed choices for completion of the selection operations of FCS_COP.1/CMAC.

Table 12: Allowed choices for FCS_COP.1/CMAC

Identifier	Cryptographic algorithm	Cryptographic key sizes	List of standards
AES-CMAC	AES using CMAC mode	256 bits	[**selection**: *ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197*] [AES]

[**selection**: *: ISO/IEC 9797-1:2011 Subclause 7.6, NIST SP 800-38B*] [CMAC]

Application Note: The use of 256-bit keys for AES algorithms is required by CNSA 1.0 and 2.0.

Evaluation Activities

FCS_COP.1/CMAC

TSS

The evaluator shall examine the TSS to verify that the IV consists of all zeros in accordance with the relevant standards.

Guidance

There is no AGD for this activity.

Tests

The following tests may 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.

AES-CMAC

Identifier	Cryptographic Algorithm	Cryptographic Key Sizes	List of Standards
AES-CMAC	AES using CMAC mode	256 bits	[**selection:** ISO/IEC 18033-3:2010 (Subclause 5.2), FIPS PUB 197] [AES] [**selection:** ISO/IEC 9797-1:2011 (Subclause 7.6), NIST SP 800-38B] [CMAC]

To test the TOE's ability to generate MAC values using AES in CMAC mode the evaluator shall perform the CMAC Generation Test and CMAC Verification Test using the following input parameters:

- *Key Size [256] bits*
- *Message size [0-524288] bits in increments of 8*
- *MAC sizes [1-128] bits*

CMAC Generation Test

The evaluator shall generate eight test cases using random keys and data for each combination of the above parameters as follows:

- *For each claimed key size,*
- *For four message sizes as follows:*

- *The smallest supported message size,*
- *The largest supported message size,*
- *Two sizes that are divisible by the block size, and*
- *Two sizes that are not divisible by the block size*

- *For three MAC sizes as follows:*
  - *The smallest supported MAC size,*
  - *The largest supported MAC size, and*
  - *Some other supported MAC size*

The evaluator shall compare the output from each test case against results generated by a known- good implementation with the same input parameters.

CMAC Verification Test

The evaluator shall generate 20 test cases using random keys and data for each combination of the above parameters as follows:

- *For each claimed key size,*
- *For four message sizes as follows:*
  - *The smallest supported message size,*
  - *The largest supported message size,*
  - *Two sizes that are divisible by the block size, and*
  - *Two sizes that are not divisible by the block size*
- *For three MAC sizes as follows:*
  - *The smallest supported MAC size,*
  - *The largest supported MAC size, and*
  - *Some other supported MAC size*

The evaluator shall modify the tag such that 25% of the test cases in each group of 20 test cases should fail.

The evaluator shall determine that the verification fails for the test cases with modified inputs, and succeeds for those with unmodified inputs.

FCS_RBG.1 Cryptographic Operation (Random Bit Generation)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_CKM.1.1/DEK,***
- ***FCS_CKM.6.2/GENHW,***
- ***FCS_CKM.6.2/SW,***
- ***FCS_CKM.6.2/TOEHW,***
- ***FCS_COP.1.1/KeyEncap,***
- ***FCS_COP.1.1/KeyEnc,***
- ***FCS_COP.1.1/KeyWrap,***
- ***FCS_CKM.5.1,***
- ***FCS_SNI_EXT.1.1***

FCS_RBG.1.1

The TSF shall perform deterministic random bit generation services using [**selection**: *Hash_DRBG (SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512), HMAC_DRBG (SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512), CTR_DRBG (AES-128, AES-192, AES-256)*] in accordance with [**selection**: *ISO/IEC 18031:2011, NIST SP 800-90A*] after initialization with a seed.

Application Note: For Hash_DRBG and HMAC_DRBG, all allowed choices support a 256-bit security strength. For CTR_DRBG, the supported security strength is equal to the AES size. The TOE is expected to use a DRBG function that can support the security strength of the keys and random values to be generated. For example, an AES-192 CTR_DRBG can be used to generate 128-bit and 192-bit symmetric keys, but can not be used to generate 256-bit symmetric keys. More information is provided in Section 8.4 of NIST SP 800-90A.

FCS_RBG.1.2

The TSF shall use a [**selection**: *TSF noise source [**assignment**: name of noise source]*, ***multiple TSF noise sources [assignment: names of noise sources]***, *TSF interface for seeding*] for initialized seeding.

Application Note: For the selection in this requirement, the ST author selects "TSF noise source" if a single noise source is used as input to the DRBG. The ST author selects "multiple TSF noise sources" if a seed is formed from a combination of two or more noise sources within the TOE boundary. If the TSF implements two or more separate DRBGs that are seeded in separate manners, this SFR should be iterated for each DRBG. It multiple distinct noise sources exist such that each DRBG only uses one of them, then each iteration would select "TSF noise source"; "multiple TSF noise sources" is only selected if a single DRBG uses multiple noise sources for its seed. The ST author selects "TSF interface for seeding" if noise source data is generated outside the TOE boundary.

If "TSF noise source" is selected, FCS_RBG.3 must be claimed.

If "multiple TSF noise sources" is selected, FCS_RBG.4 and FCS_RBG.5 must be claimed.

If "TSF interface for seeding" is selected, FCS_RBG.2 must be claimed.

FCS_RBG.1.3

The TSF shall update the RBG state by [**selection**: *reseeding, uninstantiating and reinstantiating*] using a [**selection**: *TSF noise source [**assignment**: name of noise source]*, *TSF interface for seeding*] in the following situations: [**selection**:

- *never*
- *on demand*
- *on the condition: [**assignment**: condition]*
- *after [**assignment**: time]*

] in accordance with [**assignment**: *list of standards*].

Application Note: This SFR is claimed when the TSF requires the use of random bit generation for submask generation (FCS_CKM.5.1) or salt generation (FCS_SNI_EXT.1).

Evaluation Activities

FCS_RBG.1.1
TSS
The evaluator shall verify that the TSS identifies the DRBGs used by the TOE.

Guidance
If the DRBG functionality is configurable, the evaluator shall verify that the operational guidance includes instructions on how to configure this behavior.

Tests
The evaluator shall perform the following tests:

The evaluator shall perform 15 trials for the DRBG implementation. If the DRBG is configurable, the evaluator shall perform 15 trials for each configuration. The evaluator shall also confirm that the operational guidance contains appropriate instructions for configuring the DRBG functionality.

If the DRBG 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 SP 800-90A).

If the DRBG 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 list contains 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 less than or equal to 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_RBG.1.2
Documentation will be produced - and the evaluator shall perform the activities - in accordance with Appendix D, "Entropy Documentation and Assessment" and the , Clarification to the Entropy Documentation and Assessment Annex.
TSS
There are no additional TSS required for this activity.

Guidance
There are no additional AGD required for this activity.

KMD
There are no additional KMD required for this activity.

Tests
There are no additional tests required for this activity.

FCS_RBG.1.3
TSS
The evaluator shall verify that the TSS identifies how the DRBG state is updated, and the situations under which this may occur.

Guidance
If the ST claims that the DRBG state can be updated on demand, the evaluator shall verify that the

operational guidance has instructions for how to perform this operation.

KMD
There are no additional KMD required for this activity.

Tests
There are no additional tests required for this activity.

FCS_RBG.2 Random Bit Generation (External Seeding)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_RBG.1.2***

FCS_RBG.2.1

The TSF shall be able to accept a minimum input of [**assignment**: *minimum input length greater than zero*] from a TSF interface for the purpose of seeding.

Application Note: This requirement is claimed when a DRBG is seeded with entropy from one or more noise source that is outside the TOE boundary. Typically the entropy produced by an environmental noise source is conditioned such that the input length has full entropy and is therefore usable as the seed. However, if this is not the case, it should be noted what the minimum entropy rate of the noise source is so that the TSF can collect a sufficiently large sample of noise data to be conditioned into a seed value.

Evaluation Activities

FCS_RBG.2
The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, the TSF external interface used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data such that it can be determined that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.

TSS
There are no additional TSS required for this activity.

Guidance
There are no additional AGD required for this activity.

KMD
There are no additional KMD required for this activity.

Tests
There are no additional tests required for this activity.

FCS_RBG.3 Random Bit Generation (Internal Seeding - Single Source)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_RBG.1.2***

FCS_RBG.3.1

The TSF shall be able to seed the RBG using a [**selection, choose one of**: *TSF software-based noise source*, *TSF hardware-based noise source*] [**assignment**: *name of noise source*] with a minimum of [**assignment**: *number of bits*] bits of min-entropy.

Application Note: This requirement is claimed when a DRBG is seeded with entropy from a single noise source that is within the TOE boundary. Min-entropy should be expressed as a ratio of entropy bits to sampled bits so that the total amount of data needed to ensure full entropy is known, as well as the conditioning function by which that data is reduced in size to the seed.

Evaluation Activities

FCS_RBG.3

The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, the TSF noise source used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data such that it can be determined that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.

TSS

There are no additional TSS required for this activity.

Guidance

There are no additional AGD required for this activity.

KMD

There are no additional KMD required for this activity.

Tests

There are no additional tests required for this activity.

FCS_RBG.4 Random Bit Generation (Internal Seeding - Multiple Sources)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_RBG.1.2***

FCS_RBG.4.1

The TSF shall be able to seed the RBG using [**selection**: *[**assignment**: number] TSF software-based noise sources*, *[**assignment**: number] TSF hardware-based noise*

sources].

Application Note: This requirement is claimed when a DRBG is seeded with entropy from multiple noise sources that are within the TOE boundary. FCS_RBG.5 defines the mechanism by which these sources are combined to ensure sufficient minimum entropy.

Evaluation Activities

FCS_RBG.4

The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, each TSF noise source used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data from each data source.

TSS

There are no additional TSS required for this activity.

Guidance

There are no additional AGD required for this activity.

KMD

There are no additional KMD required for this activity.

Tests

There are no additional tests required for this activity.

FCS_RBG.5 Random Bit Generation (Combining Noise Sources)

The inclusion of this selection-based component depends upon selection in:

- ***FCS_RBG.1.2***

FCS_RBG.5.1

The TSF shall [**assignment**: *combining operation*] [**selection**: *output from TSF noise sources, input from TSF interfaces for seeding)*] to create the entropy input into the derivation function as defined in [**assignment**: *list of standards*], resulting in a minimum of [**assignment**: *number of bits*] bits of min-entropy.

Application Note: Examples of typical combining operations include, but are not limited to, XORing or hashing.

Evaluation Activities

FCS_RBG.5

Using the entropy sources specified in FCS_RBG.4, the evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it describes the method by which the various entropy sources are combined into a single seed. This should include an estimation of the rate at which each noise source outputs data and whether this is dependent on any system-specific factors so that each source's relative contribution to the overall entropy is understood. The evaluator shall verify that

the resulting combination of sampled data and the min-entropy rate of the sampled data is described in sufficient detail to determine that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.

TSS
There are no additional TSS required for this activity.

Guidance
There are no additional AGD required for this activity.

KMD
There are no additional KMD required for this activity.

Tests
There are no additional tests required for this activity.

FCS_SMC_EXT.1 Submask Combining

The inclusion of this selection-based component depends upon selection in:

- ***FCS_KYC_EXT.2.2,***
- ***FPT_KYP_EXT.1.1***

FCS_SMC_EXT.1.1

The TSF shall combine submasks using the following method [**selection**: *exclusive OR (XOR), SHA-256, SHA-384, SHA-512*] to generate an [*intermediary key*].

Application Note: This requirement specifies the way that a product may combine the various submasks by using either an XOR or an approved SHA-hash. The approved hash functions are captured in FCS_COP.1/Hash.

This SFR is claimed when the TSF requires the use of submask combining as part of maintaining or deriving a key chain.

Evaluation Activities

FCS_SMC_EXT.1
TSS
If the submasks produced from the authorization factors are XORed together to form the BEV or intermediate key, the TSS section shall identify how this is performed (e.g., if there are ordering requirements, checks performed, etc.). The evaluator shall also confirm that the TSS describes how the length of the output produced is at least the same as that of the BEV.

Guidance
There are no AGD evaluation activities for this SFR.

KMD

The evaluator shall review the KMD to ensure that an approved combination is used and does not result in the weakening or exposure of key material.

Tests
The evaluator shall perform the following test:

- *Test FCS_SMC_EXT.1:1: (conditional): If there is more than one authorization factor, ensure that failure to supply a required authorization factor does not result in access to the encrypted data.*

B.2 Protection of the TSF (FPT)

FPT_FLS.1 Failure with Preservation of Secure State

The inclusion of this selection-based component depends upon selection in:

- ***FCS_RBG.1.2***

FPT_FLS.1.1

The TSF shall preserve a secure state when the following types of failures occur: [*DRBG self-test failure*].

Application Note: This requirement must be included if FCS_RBG.1 is included in the ST. The intent of this requirement is to ensure that cryptographic services requiring random bit generation cannot be performed if a failure of a self-test defined in FPT_TST.1 occurs.

Evaluation Activities

FPT_FLS.1
TSS
The evaluator shall verify that the TSF describes how the TOE enters an error state in the event of a DRBG self-test failure.

Guidance
The evaluator shall verify that the guidance documentation describes the error state that results from a DRBG self-test failure and the actions that a user or administrator should take in response to attempt to resolve the error state.

Tests
There is no Test for this activity.

FPT_FUA_EXT.1 Firmware Update Authentication

The inclusion of this selection-based component depends upon selection in:

- ***FPT_TUD_EXT.1.3***

FPT_FUA_EXT.1.1

The TSF shall authenticate the source of the firmware update using the digital signature algorithm specified in FCS_COP.1/SigVer using the RTU that contains [**selection**: *the public key, hash value of the public key as specified in FCS_COP.1/Hash*]

FPT_FUA_EXT.1.2

The TSF shall only allow installation of update if the digital signature has been successfully verified as specified in FCS_COP.1/SigVer.

FPT_FUA_EXT.1.3

The TSF shall only allow modification of the existing firmware after the successful validation of the digital signature, using a mechanism as described in FPT_TUD_EXT.1.2.

Application Note: The firmware portion of the TOE (e.g., RTU (key store and the signature verification algorithm)) is expected to be stored in a write protected area on the TOE. It is expected that the firmware only be modifiable in a post-manufacturing state using the authenticated update mechanism described in FPT_FUA_EXT.1. The TSF is modifiable only by using the mechanisms specified in FPT_TUD_EXT.1.

FPT_FUA_EXT.1.4

The TSF shall return an error code if any part of the firmware update process fails.

Application Note: This SFR must be claimed if "authenticated firmware update mechanism as described in FPT_FUA_EXT.1" is claimed in FPT_TUD_EXT.1.3.

The authenticated firmware update mechanism employs digital signatures to ensure the authenticity of the firmware update image. The TSF provides a RTU that contains a signature verification algorithm and a key store that includes the public key needed to verify the signature on the update image. The key store in the RTU should include a public key used to verify the signature on an update image or a hash of the public key if a copy of the public key is provided with the update image. In the latter case, the update mechanism should 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 on 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.

The intent of this requirement is to specify that the authenticated update mechanism should ensure that the new image has been digitally signed; and that the digital signature can be verified by using a public key before the update takes place. The requirement also specifies that the authenticated update mechanism only allows installation of updates when the digital signature has been successfully verified by the TSF.

Evaluation Activities

FPT_FUA_EXT.1

TSS

The evaluator shall examine the TSS to ensure that it describes how the TOE uses the RTU, what type of key or hash value, and where the value is stored on the RTU. The evaluator shall also verify that the TSS contains a description (storage location) of where the original firmware exists.

Guidance

There is no AGD for this activity.

KMD
There is no KMD for this activity.

Tests
There is no test for this activity

Appendix C - Extended Component Definitions

This appendix contains the definitions for all extended requirements specified in the PP.

C.1 Extended Components Table

All extended components specified in the PP are listed in this table:

Table 13: Extended Component Definitions

Functional Class	Functional Components
Cryptographic Support (FCS)	FCS_CKM_EXT Cryptographic Key Destruction Types FCS_KYC_EXT Key Chaining FCS_SMC_EXT Submask Combining FCS_SNI_EXT Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation) FCS_VAL_EXT Validation of Cryptographic Elements
Protection of the TSF (FPT)	FPT_FAC_EXT Firmware / Software Access Control FPT_FUA_EXT Firmware Update Authentication FPT_KYP_EXT Key and Key Material Protection FPT_PWR_EXT Power Management FPT_RBP_EXT Rollback Protection FPT_TUD_EXT Trusted Update
User Data Protection	FDP_DSK_EXT Protection of Data on Disk

C.2 Extended Component Definitions

C.2.1 Cryptographic Support (FCS)

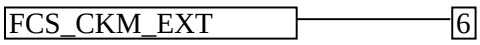
This PP defines the following extended components as part of the FCS class originally defined by CC Part 2:

C.2.1.1 FCS_CKM_EXT Cryptographic Key Destruction Types

Family Behavior

This family is intended to support the ability to specify the implementation of multiple key destruction methods.

Component Leveling



FCS_CKM_EXT.6, Cryptographic Key Destruction Types, provides the TOE with the ability to select between multiple methods of key destruction.

Management: FCS_CKM_EXT.6

There are no management functions foreseen.

Audit: FCS_CKM_EXT.6

There are no audit events foreseen.

FCS_CKM_EXT.6 Cryptographic Key Destruction Types

Hierarchical to: No other components.

Dependencies to: FCS_CKM.6 Cryptographic Key and Key Material Destruction

FCS_CKM_EXT.6.1

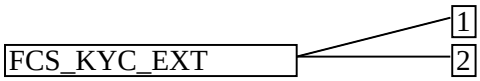
The TSF shall use [**assignment**: *one or more iterations of FCS_CKM.6 defined elsewhere in the Security Target*] key destruction methods.

C.2.1.2 FCS_KYC_EXT Key Chaining

Family Behavior

This family provides the specification to be used for using multiple layers of encryption keys to ultimately secure the protected data encrypted on the drive.

Component Leveling



FCS_KYC_EXT.1, Key Chaining (Initiator), requires the TSF to maintain a key chain for a BEV that is provided to a component external to the TOE. Note that this cPP does not include FCS_KYC_EXT.1; it is only included here to provide a complete definition of the FCS_KYC_EXT family.

FCS_KYC_EXT.2, Key Chaining (Recipient), requires the TSF to be able to accept a BEV that is then chained to a DEK used by the TSF through some method.

Management: FCS_KYC_EXT.1

There are no management functions foreseen.

Audit: FCS_KYC_EXT.1

There are no audit events foreseen.

FCS_KYC_EXT.1 Key Chaining (Initiator)

Hierarchical to: No other components.

Dependencies to: FCS_CKM.1 Cryptographic Key Generation

FCS_COP.1 Cryptographic Operation

FCS_CKM.5 Cryptographic Key Derivation

FCS_SMC_EXT.1 Submask Combining

FCS_VAL_EXT.1 Validation

FCS_KYC_EXT.1.1

The TSF shall maintain a key chain of: [**selection**:

- *one, using a submask as the BEV;*
- *intermediate keys originating from one or more submasks to the BEV using the following methods: [**selection**:*
  - *key encryption as specified in FCS_COP.1*
  - *key transport as specified in FCS_COP.1,*
  - *key wrapping as specified in FCS_COP.1,*
  - *key derivation as specified in FCS_CKM.5*
  - *key combining as specified in FCS_SMC_EXT.1,*

  ]

] while maintaining an effective strength of [**selection**: *128 bits*, *256 bits*] for symmetric keys and an effective strength of [**selection**: *not applicable*, *112 bits*, *128 bits*, *192 bits*, *256 bits*] for asymmetric keys.

FCS_KYC_EXT.1.2

The TSF shall provide at least a [**selection**: *128 bits*, *256 bits*] BEV to [**assignment**: *one or more external entities*] [**selection**:

- *after the TSF has successfully performed the validation process as specified in FCS_VAL_EXT.1*
- *without validation taking place*

]

Management: FCS_KYC_EXT.2

There are no management functions foreseen.

Audit: FCS_KYC_EXT.2

There are no audit events foreseen.

FCS_KYC_EXT.2 Key Chaining (Recipient)

Hierarchical to:	No other components.
Dependencies to:	FCS_CKM.1 Cryptographic Key Generation
	FCS_COP.1 Cryptographic Operation
	FCS_CKM.5 Cryptographic Key Derivation
	FCS_SMC_EXT.1 Submask Combining

FCS_KYC_EXT.2.1

The TSF shall accept a BEV of at least 256 bits.

FCS_KYC_EXT.2.2

The TSF shall maintain a chain of intermediary keys originating from the BEV to the DEK using the following methods: [**selection**:

- *key derivation as specified in FCS_CKM.5*
- *key wrapping as specified in FCS_COP.1*
- *key encryption as specified in FCS_COP.1*
- *key transport as specified in FCS_COP.1*
- *key combining as specified in FCS_SMC_EXT.1*

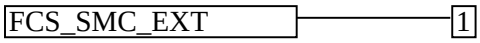
] while maintaining an effective strength of [256 bits] for symmetric keys and an effective strength of [**selection**: *not applicable, 128 bits, 192 bits, 256 bits*] for asymmetric keys.

C.2.1.3 FCS_SMC_EXT Submask Combining

Family Behavior

This family specifies the means by which submasks are combined, if the TOE supports more than one submask being used to derive or protect the BEV.

Component Leveling



FCS_SMC_EXT.1, Submask Combining, requires the TSF to combine the submasks in a predictable fashion.

Management: FCS_SMC_EXT.1

There are no management functions foreseen.

Audit: FCS_SMC_EXT.1

There are no audit events foreseen.

FCS_SMC_EXT.1 Submask Combining

Hierarchical to: No other components.

Dependencies to: FCS_COP.1 Cryptographic Operation

FCS_SMC_EXT.1.1

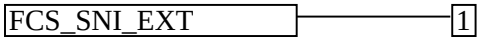
The TSF shall combine submasks using the following method [**selection**: *exclusive OR (XOR), SHA-256, SHA-384, SHA-512*] to generate an [**assignment**: *types of keys*].

C.2.1.4 FCS_SNI_EXT Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation)

Family Behavior

This family ensures that salts, nonces, and IVs are well formed.

Component Leveling



FCS_SNI_EXT.1, Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation), requires the generation of salts, nonces, and IVs to be used by the cryptographic components of the TOE to be performed in the specified manner.

Management: FCS_SNI_EXT.1

There are no management functions foreseen.

Audit: FCS_SNI_EXT.1

There are no audit events foreseen.

FCS_SNI_EXT.1 Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation)

Hierarchical to: No other components.

Dependencies to: FCS_RBG.1 Cryptographic Operation (Random Bit Generation)

FCS_SNI_EXT.1.1

The TSF shall [**selection**:

- *use no salts*
- *use salts that are generated by a [**selection**: DRBG as specified in FCS_RBG.1, DRBG provided by the OE]*

].

FCS_SNI_EXT.1.2

The TSF shall use [**selection**: *no nonces, unique nonces with a minimum size of [**assignment**: number of bits] bits*].

FCS_SNI_EXT.1.3

The TSF shall [**selection**:

- *use no IVs*
- *create IVs in the following manner [**selection**:*
  - *CBC: IVs shall be non-repeating and unpredictable*
  - *CCM: Nonce shall be non-repeating and unpredictable*
  - *XTS: No IV. Tweak values shall be non-negative integers, assigned consecutively, and starting at an arbitrary non-negative integer;*
  - *GCM: IV shall be non-repeating. The number of invocations of GCM shall not exceed 2^32 for a given secret key*

  ]

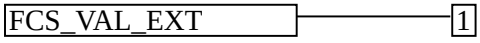
].

C.2.1.5 FCS_VAL_EXT Validation of Cryptographic Elements

Family Behavior

This family specifies the means by which submasks and/or BEVs are determined to be valid prior to their use.

Component Leveling



FCS_VAL_EXT.1, Validation, requires the TSF to validate submasks and BEVs by one or more of the specified methods.

Management: FCS_VAL_EXT.1

There are no management functions foreseen.

Audit: FCS_VAL_EXT.1

There are no audit events foreseen.

FCS_VAL_EXT.1 Validation

Hierarchical to:	No other components.
Dependencies to:	FCS_COP.1 Cryptographic Operation

FCS_VAL_EXT.1.1

The TSF shall perform validation of the [**selection**: *submask, intermediate key, BEV*] using the following methods: [**selection**:

- *key wrap as specified in FCS_COP.1;*
- *hash the [**selection**: submask, intermediate key, BEV] as specified in [**assignment**: cryptographic operation requirement] and compare it to a stored hashed [**selection**: submask, intermediate key, BEV];*
- *decrypt a known value using the [**selection**: submask, intermediate key, BEV] specified in FCS_COP.1 and compare it against a stored known value*

].

FCS_VAL_EXT.1.2

The TSF shall require validation of the [**selection**: *submask, intermediate key, BEV*] prior to [**assignment**: *activity requiring validation*].

FCS_VAL_EXT.1.3

The TSF shall [**selection**:

- *perform a key sanitization of the DEK upon a [**selection**: configurable number, [**assignment**: ST author specified number]] of consecutive failed validation attempts*
- *institute a delay such that only [**assignment**: ST author specified number of attempts] can be made within a 24 hour period*
- *block validation after [**assignment**: ST author specified number of attempts] of consecutive failed validation attempts*
- *require power cycle or reset of the TOE after [**assignment**: ST author specified number of attempts] of consecutive failed validation attempts*

].

C.2.2 Protection of the TSF (FPT)

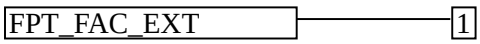
This PP defines the following extended components as part of the FPT class originally defined by CC Part 2:

C.2.2.1 FPT_FAC_EXT Firmware / Software Access Control

Family Behavior

This family requires that a valid authentication factor be provided prior to the TSF authorizing an update.

Component Leveling



FPT_FAC_EXT.1, Firmware / Software Access Control, requires the TSF to require an authentication factor or action prior to allowing an update to be performed.

Management: FPT_FAC_EXT.1

The following actions could be considered for the management functions in FMT:

- management of the password used to authorize the update

Audit: FPT_FAC_EXT.1

There are no audit events foreseen.

FPT_FAC_EXT.1 Firmware / Software Access Control

Hierarchical to: No other components.

Dependencies to: No dependencies.

FPT_FAC_EXT.1.1

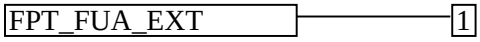
The TSF shall require [**selection**: *a password*, *a unique value printed on the device*, *a authorized user action*] before the [**selection**: *software*, *firmware*] update proceeds.

C.2.2.2 FPT_FUA_EXT Firmware Update Authentication

Family Behavior

This family requires that firmware updates be authenticated by the TSF prior to being applied.

Component Leveling



FPT_FUA_EXT.1, Firmware Update Authentication, requires the TSF to authenticate firmware updates using a specified method.

Management: FPT_FUA_EXT.1

There are no management functions foreseen.

Audit: FPT_FUA_EXT.1

There are no audit events foreseen.

FPT_FUA_EXT.1 Firmware Update Authentication

Hierarchical to: No other components.

Dependencies to: FCS_COP.1 Cryptographic Operation

FPT_FUA_EXT.1.1

The TSF shall authenticate the source of the firmware update using the digital signature algorithm specified in FCS_COP.1 using the RTU that contains [**selection**: *the public key*, *hash value of the public key as specified in FCS_COP.1*]

FPT_FUA_EXT.1.2

The TSF shall only allow installation of update if the digital signature has been successfully verified as specified in FCS_COP.1.

FPT_FUA_EXT.1.3

The TSF shall only allow modification of the existing firmware after the successful validation of the digital signature, using a mechanism as described in FPT_TUD_EXT.1.2.

FPT_FUA_EXT.1.4

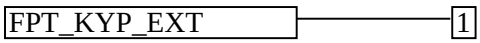
The TSF shall return an error code if any part of the firmware update process fails.

C.2.2.3 FPT_KYP_EXT Key and Key Material Protection

Family Behavior

This family requires that key and key material be protected if and when written to non-volatile storage.

Component Leveling



FPT_KYP_EXT.1, Protection of Key and Key Material, requires the TSF to ensure that no plaintext key or key material are written to non-volatile storage.

Management: FPT_KYP_EXT.1

There are no management functions foreseen.

Audit: FPT_KYP_EXT.1

There are no audit events foreseen.

FPT_KYP_EXT.1 Protection of Key and Key Material

Hierarchical to:	No other components.
Dependencies to:	FCS_COP.1 Cryptographic Operation
	FCS_KYC_EXT.1 Key Chaining (Initiator)
	FCS_KYC_EXT.2 Key Chaining (Recipient)
	FCS_SMC_EXT.1 Submask Combining

FPT_KYP_EXT.1.1

The TSF shall [**selection**:

- *not store keys in non-volatile memory*

- *only store keys in non-volatile memory when wrapped, as specified in FCS_COP.1, or encrypted, as specified in FCS_COP.1*
- *only store plaintext keys that meet any one of the following criteria [**selection**:*
  - *the plaintext key is not part of the key chain as specified in FCS_KYC_EXT.2*
  - *the plaintext key will no longer provide access to the encrypted data after initial provisioning*
  - *the plaintext key is a key split that is combined as specified in FCS_SMC_EXT.1, and the other half of the key split is [**selection**:*
    - *wrapped as specified in FCS_COP.1*
    - *encrypted as specified in FCS_COP.1*
    - *derived and not stored in non-volatile memory*

    ]
  - *the non-volatile memory the key is stored on is located in an external storage device for use as an authorization factor*
  - *the plaintext key is only used to provide additional cryptographic protection to other keys, such that disclosure of the plaintext key would not compromise the security of the keys being protected*

]

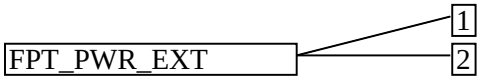
].

C.2.2.4 FPT_PWR_EXT Power Management

Family Behavior

This family defines secure behavior of the TSF when the TOE supports multiple power saving states. The use of compliant power saving states (i.e. power saving states that purge security relevant data upon entry) is essential for ensuring that state transitions cannot be used as attack vectors to bypass TOE self-protection mechanisms.

Component Leveling



FPT_PWR_EXT.1, Power Saving States, defines the compliant power saving states that are implemented by the TSF.

FPT_PWR_EXT.2, Timing of Power Saving States, describes the situations that cause compliant power saving states to be entered.

Management: FPT_PWR_EXT.1

The following actions could be considered for the management functions in FMT:

- Enable or disable the use of individual power saving states
- Specify one or more power saving state configurations

Audit: FPT_PWR_EXT.1

There are no auditable events foreseen.

FPT_PWR_EXT.1 Power Saving States

Hierarchical to: No other components.

Dependencies to: No dependencies

FPT_PWR_EXT.1.1

The TSF shall define the following compliant power saving states: [**selection**: *S3*, *S4*, *G2(S5)*, *G3*, *D0*, *D1*, *D2*, *D3*, *[**assignment**: other power saving states]*].

Management: FPT_PWR_EXT.2

There are no management functions foreseen.

Audit: FPT_PWR_EXT.2

The following actions should be auditable if FAU_GEN Security audit data generation is included in the cPP/ST:

- Transition of the TSF into different power saving states

FPT_PWR_EXT.2 Timing of Power Saving States

Hierarchical to: No other components.

Dependencies to: FPT_PWR_EXT.1 Power Saving States

FPT_PWR_EXT.2.1

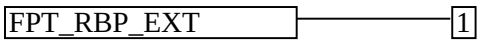
For each compliant power saving state defined in FPT_PWR_EXT.1.1, the TSF shall enter the compliant power saving state when the following conditions occur: user-initiated request, [**selection**: *shutdown*, *user inactivity*, *request initiated by remote management system*, *[**assignment**: other conditions]*, *no other conditions*].

C.2.2.5 FPT_RBP_EXT Rollback Protection

Family Behavior

This family requires that the TSF protects against rollbacks or downgrades to its version.

Component Leveling



FPT_RBP_EXT.1, Rollback Protection, requires the TSF to detect and prevent unauthorized rollback.

Management: FPT_RBP_EXT.1

There are no management functions foreseen.

Audit: FPT_RBP_EXT.1

There are no audit events foreseen.

FPT_RBP_EXT.1 Rollback Protection

Hierarchical to: No other components.

Dependencies to: No dependencies.

FPT_RBP_EXT.1.1

The TSF shall verify that the new update is not downgrading to a lower security version number by [**assignment**: *method of verifying the security version number is the same as or higher than the currently installed version*].

FPT_RBP_EXT.1.2

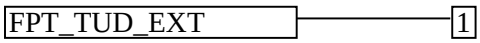
The TSF shall generate and return an error code if the attempted update package is detected to be an invalid version.

C.2.2.6 FPT_TUD_EXT Trusted Update

Family Behavior

Components in this family address the requirements for updating the TOE firmware / software.

Component Leveling



FPT_TUD_EXT.1, Trusted Update, requires the capability to be provided to update the TOE firmware and software, including the ability to verify the updates prior to installation.

Management: FPT_TUD_EXT.1

The following actions could be considered for the management functions in FMT:

- Ability to update the TOE and to verify the updates

Audit: FPT_TUD_EXT.1

The following actions should be auditable if FAU_GEN Security audit data generation is included in the cPP/ST:

- Initiation of the update process
- Any failure to verify the integrity of the update

FPT_TUD_EXT.1 Trusted Update

Hierarchical to: No other components.

Dependencies to: FCS_COP.1 Cryptographic Operation

FPT_TUD_EXT.1.1

The TSF shall provide [**assignment**: *list of subjects*] the ability to query the current version of the TOE [**selection**: *software*, *firmware*].

FPT_TUD_EXT.1.2

The TSF shall provide [**assignment**: *list of subjects*] the ability to initiate updates to TOE [**selection**: *software*, *firmware*].

FPT_TUD_EXT.1.3

The TSF shall verify updates to the TOE software using a [**selection**: *digital signature*, *published hash*] by the manufacturer prior to installing those updates.

C.2.3 User Data Protection

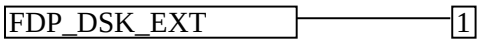
This PP defines the following extended components as part of the class originally defined by CC Part 2:

C.2.3.1 FDP_DSK_EXT Protection of Data on Disk

Family Behavior

This family specifies methods for ensuring that data residing in permanent storage on disk is not subject to unauthorized disclosure.

Component Leveling



FDP_DSK_EXT.1, Protection of Data on Disk, requires the TSF to encrypt all protected data without user intervention using full drive encryption.

Management: FDP_DSK_EXT.1

There are no management functions foreseen.

Audit: FDP_DSK_EXT.1

There are no audit events foreseen.

FDP_DSK_EXT.1 Protection of Data on Disk

Hierarchical to: No other components.

Dependencies to: FCS_COP.1 Cryptographic Operation

FDP_DSK_EXT.1.1

The TSF shall perform Full Drive Encryption in accordance with FCS_COP.1, such that the drive contains no plaintext protected data.

FDP_DSK_EXT.1.2

The TSF shall encrypt all protected data without user intervention.

Appendix D - Entropy Documentation and Assessment

This is an optional appendix in the cPP, and only applies if the TOE is providing deterministic random bit generation services, e.g. the ST claims FCS_RBG.1.

This appendix describes the required supplementary information for each entropy source used by the TOE.

The documentation of the entropy sources should be detailed enough that, after reading, the evaluator will thoroughly understand the entropy source and why it can be relied upon to provide sufficient entropy. This documentation should include multiple detailed sections: design description, entropy justification, operating conditions, and health testing. This documentation is not required to be part of the TSS in the public facing ST.

D.1 Design Description

Documentation shall include the design of each entropy source as a whole, including the interaction of all entropy source components. Any information that can be shared regarding the design should also be included for any third-party entropy sources that are included in the product.

The documentation will describe the operation of the entropy source to include how entropy is produced, and how unprocessed (raw) data can be obtained from within the entropy source for testing purposes. The documentation should walk through the entropy source design indicating where the entropy comes from, where the entropy output is passed next, any post-processing of the raw outputs (hash, XOR, etc.), if/where it is stored, and finally, how it is output from the entropy source. Any conditions placed on the process (e.g., blocking) should also be described in the entropy source design. Diagrams and examples are encouraged.

This design must also include a description of the content of the security boundary of the entropy source and a description of how the security boundary ensures that an adversary outside the boundary cannot affect the entropy rate.

If implemented, the design description shall include a description of how third-party applications can add entropy to the RBG. A description of any RBG state saving between power-off and power-on shall be included.

D.2 Entropy Justification

There should be a technical argument for where the unpredictability in the source comes from and why there is confidence in the entropy source delivering sufficient entropy for the uses made of the RBG output (by this particular TOE). This argument will include a description of the expected min-entropy rate (i.e. the minimum entropy (in bits) per bit or byte of source data) and explain that sufficient entropy is going into the TOE randomizer seeding process. This discussion will be part of a justification for why the entropy source can be relied upon to produce bits with entropy.

The amount of information necessary to justify the expected min-entropy rate depends on the type of entropy source included in the product.

For developer provided entropy sources, in order to justify the min-entropy rate, it is expected that a large number of raw source bits will be collected, statistical tests will be performed, and the min-entropy rate determined from the statistical tests. While no particular statistical tests are required at this time, it is expected that some testing is necessary in order to determine the amount of min-entropy in each output.

For third party provided entropy sources, in which the TOE vendor has limited access to the design and raw entropy data of the source, the documentation will indicate an estimate of the amount of min-entropy obtained from this third-party source. It is acceptable for the vendor to “assume” an amount of min-entropy, however, this assumption must be clearly stated in the documentation provided. In particular, the min-entropy estimate must be specified and the assumption included in the ST.

Regardless of type of entropy source, the justification will also include how the DRBG is initialized with the entropy stated in the ST, for example by verifying that the min-entropy rate is multiplied by the amount of source data used to seed the DRBG or that the rate of entropy expected based on the amount of source data is explicitly stated and compared to the statistical rate. If the amount of source data used to seed the DRBG is not clear or the calculated rate is not explicitly related to the seed, the documentation will not be considered complete.

The entropy justification shall not include any data added from any third-party application or from any state saving between restarts.

D.3 Operating Conditions

The entropy rate may be affected by conditions outside the control of the entropy source itself. For example, voltage, frequency, temperature, and elapsed time after power-on are just a few of the factors that may affect the operation of the entropy source. As such, documentation will also include the range of operating conditions under which the entropy source is expected to generate random data. Similarly, documentation shall describe the conditions under which the entropy source is no longer guaranteed to provide sufficient entropy. Methods used to detect failure or degradation of the source shall be included.

D.4 Health Testing

More specifically, all entropy source health tests and their rationale will be documented. This will include a description of the health tests, the rate and conditions under which each health test is performed (e.g., at startup, continuously, or on-demand), the expected results for each health test, TOE behavior upon entropy source failure, and rationale indicating why each test is believed to be appropriate for detecting one or more failures in the entropy source.

Appendix E - Key Management Description

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.

Essay:

The essay will provide the following information for all keys in the key chain:

- The purpose of the key
- If the key is stored in non-volatile memory
- How and when the key is protected
- How and when the key is derived
- The strength of the key
- When or if the key would be no longer needed, along with a justification.

The essay will also describe the following topics:

- A description of all authorization factors that are supported by the product and how each factor is handled, including any conditioning and combining performed.
- If validation is supported, the process for validation shall be described, noting what value is used for validation and the process used to perform the validation. It shall describe how this process ensures no keys in the key chain are weakened or exposed by this process.
- The authorization process that leads to the ultimate release of the BEV. This section shall detail the key chain used by the product. It shall describe which keys are used in the protection of the BEV and how they meet the derivation, key wrap, or a combination of the two requirements, including the direct chain from the initial authorization to the BEV. It shall also include any values that add into that key chain or interact with the key chain and the protections that ensure those values do not weaken or expose the overall strength of the key chain.
- The diagram and essay will clearly illustrate the key hierarchy to ensure that at no point the chain could be broken without a cryptographic exhaust or all of the initial authorization values and the effective strength of the BEV is maintained throughout the Key Chain.
- A description of the data encryption engine, its components, and details about its implementation (e.g. for hardware: integrated within the device's main SOC or separate co-processor, for software: initialization of the product, drivers, libraries (if applicable), logical interfaces for encryption/decryption, and areas which are not encrypted (e.g. boot loaders, portions associated with the Master Boot Record (MBRs), partition tables, etc.)). The description should also include the data flow from the device's host interface to the device's persistent media storing the data, information on those conditions in which the data bypasses the data encryption engine (e.g. read-write operations to an unencrypted Master Boot Record area). The description should be detailed enough to verify all platforms to ensure that when the user enables encryption, the product encrypts all hard storage devices. It should also describe the platform's boot initialization, the encryption initialization process, and at what moment the product enables the encryption.
- The process for destroying keys when they are no longer needed by describing the storage location of all keys and the protection of all keys stored in non-volatile memory.

Diagram:

- The diagram will include all keys from the initial authorization factors to the BEV and any keys or values that contribute into the chain. It must list the cryptographic strength of each key and indicate how each key along the chain is protected with either key derivation or key wrapping (from the allowed options). The diagram should indicate the input used to derive or unwrap each key in the chain.
- A functional (block) diagram showing the main components (such as memories and processors) and the data path between, for hardware, the device's host interface and the device's persistent media storing the data, or for

software, the initial steps needed for the activities the TOE performs to ensure it encrypts the storage device entirely when a user or administrator first provisions the product. The hardware encryption diagram shall show the location of the data encryption engine within the data path.

- The hardware encryption diagram shall show the location of the data encryption engine within the data path. The evaluator shall validate that the hardware encryption diagram contains enough detail showing the main components within the data path and that it clearly identifies the data encryption engine.

Appendix F - Acronyms

Table 14: Acronyms

Acronym	Meaning
AA	Authorization Acquisition
AES	Advanced Encryption Standard
Base-PP	Base Protection Profile
BEV	Border Encryption Value
BIOS	Basic Input Output System
CBC	Cipher Block Chaining
CC	Common Criteria
CCM	Counter with CBC-Message Authentication Code
CEM	Common Evaluation Methodology
cPP	Collaborative Protection Profile
DEK	Data Encryption Key
DRBG	Deterministic Random Bit Generator
DSS	Digital Signature Standard
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EE	Encryption Engine
EEPROM	Electrically Erasable Programmable Read-Only Memory
EP	Extended Package
FDE	Full Drive Encryption
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards
FP	Functional Package
GCM	Galois Counter Mode
HMAC	Keyed-Hash Message Authentication Code

HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
IT	Information Technology
ITSEF	IT Security Evaluation Facility
IV	Initialization Vector
KEK	Key Encryption Key
KMD	Key Management Description
KRK	Key Release Key
MBR	Master Boot Record
NIST	National Institute of Standards and Technology
OE	Operational Environment
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PP	Protection Profile
PP-Configuration	Protection Profile Configuration
PP-Module	Protection Profile Module
PRF	Pseudo Random Function
RBG	Random Bit Generator
RNG	Random Number Generator
RoT	Root of Trust
RSA	Rivest Shamir Adleman Algorithm
RTU	Root of Trust for Update
SAR	Security Assurance Requirement
SED	Self-Encrypting Drive
SFR	Security Functional Requirement
SHA	Secure Hash Algorithm
SPD	Security Problem Definition
SPI	Serial Peripheral Interface

ST	Security Target
TOE	Target of Evaluation
TPM	Trusted Platform Module
TSF	TOE Security Functionality
TSF	TOE Security Functionality
TSFI	TSF Interface
TSS	TOE Summary Specification
TSS	TOE Summary Specification
USB	Universal Serial Bus
XOR	Exclusive or
XTS	XEX (XOR Encrypt XOR) Tweakable Block Cipher with Ciphertext Stealing

Appendix G - Bibliography

Table 15: Bibliography

Identifier	Title
[CC]	Common Criteria for Information Technology Security Evaluation - • Part 1: Introduction and general model, CCMB-2022-11-001, CC:2022, Revision 1, November 2022. • Part 2: Security functional requirements, CCMB-2022-11-002, CC:2022, Revision 1, November 2022. • Part 3: Security assurance requirements, CCMB-2022-11-003, CC:2022, Revision 1, November 2022. • Part 4: Framework for the specification of evaluation methods and activities, CCMB-2022-11-004, CC:2022, Revision 1, November 2022. • Part 5: Pre-defined packages of security requirements, CCMB-2022-11-005, CC:2022, Revision 1, November 2022.
[CEM]	Common Methodology for Information Technology Security Evaluation - • Evaluation methodology, CCMB-2022-11-006, CC:2022, Revision 1, November 2022.
[FDE–EE]	collaborative Protection Profile for Full Drive Encryption – Encryption Engine, Version 3.0, March, 2026